Global Optimization

Theory Versus Practice

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Doctor of Philosophy in Mathematics

in the

University of Canterbury

by

Chris Stephens

University of Canterbury

1997

Abstract

This thesis looks at some theoretical and practical aspects of global optimization—as we shall see they do not always coincide.

Chapter 1 defines the global optimization problem, discusses applications, concluding there are fewer than often claimed, and a presents a survey of algorithms. A simple deterministic analogue to PAS, a theoretical stochastic algorithm known to have good convergence properties, is presented. The often-claimed minimax optimality of the Piyavskii-Shubert algorithm is shown not to apply for the first few iterations. The counter-example given also applies to Mladineo's algorithm.

Chapter 2 concentrates on some theoretical results for global optimization algorithms. The results show that for both deterministic and stochastic algorithms, global information about the function is necessary to solve the global optimization problem.

Chapter 3 introduces interval arithmetic as a tool for global optimization. A simpler and slightly more general proof of the convergence of the natural inclusion function than appears in the literature is provided. Interval arithmetic is generalised to apply to new classes of subdomains and take account of the structure of the function's expression. Examples show that generalised interval arithmetic can lead to dramatic improvements in inclusions and global optimization algorithms.

Chapter 4 defines interval and bounding Hessians. The main result provides an optimal method of obtaining optimal (in two different senses) bounding Hessians from interval Hessians. Examples demonstrate the usefulness of bounding Hessians to global optimization.

Chapter 5 brings together the theoretical results of the previous chapters into a new adaptive second derivative branch and bound algorithm. First, it presents a summary of the branch and bound framework and discusses the algorithms of Baritompa and Cutler. A counter-example shows that one of Baritompa and Cutler's algorithms is not valid in general and restricted sufficient conditions under which it is valid are given. The new algorithm is based (somewhat loosely in its final form) on Baritompa and Cutler's algorithms in a branch and bound framework. It achieves for the first time a cubic order of convergence in the bounding rule of a global optimization algorithm. Theoretical implications of a cubic order of convergence

are also presented.

Chapter 6 presents the results of testing an implementation of the new algorithm and variations. Four different bounding rules, three using adaptive second derivatives, are compared on 29 test functions.

Conclusions are presented in the final chapter.

Statement of originality

Chapter 2 of this thesis is based on joint work with my Supervisor, Bill Baritompa, and has been accepted for publication in the *Journal of Optimization Theory and Applications*. My contribution to this chapter includes the stochastic results and proofs.

Chapter 4 of this thesis is based on my own paper [66] published in *Developments in Global Optimization* [7].

The remaining body of work is the product of my own scholarship and research and has not previously been submitted for assessment here or elsewhere.

Chris Stephens

Acknowledgements

It has been a pleasure to have Bill Baritompa as my supervisor and I thank him for all his advice over the years. I hope that I have returned as much as he has given me.

I thank those people whom proof-read my thesis: David Bryant, my father and especially Bill, who took the full frontal assault of my atrocious spelling and grammar.

My thanks also to Chris Tuffley for providing me with an interesting global optimization problem and enjoyable hours discussing this and other things.

I owe a debt of gratitude (as well as monetary one) to my parents for their financial understanding over the years.

Finally, I thank Peter Renaud and the Department, Ernie Tuck of the Department of Applied Mathematics at the University of Adelaide, the New Zealand Mathematics Society, the New Zealand Royal Society, my parents and the financial committee of the Third Workshop on Global Optimization for their financial support which allowed me to attend the Third Workshop on Global Optimization.

Contents

Chapter 1

Introduction

1.1 Global Optimization

This thesis deals with the theory and practice of *global optimization*. The problem of global optimization has a deceptively simple mathematical description: given a real valued *objective function* $f : D_0 \to \mathbb{R}$, where $D_0 \subseteq \mathbb{R}^n$, find a function value y^* such that $y^* \geq f(\mathbf{x})$ for all $\mathbf{x} \in D_0$. The function value y^* is called the *global maximum*. A point $\mathbf{x}^* \in D_0$ such that $f(\mathbf{x}^*) = y^*$ is called a *global maximizer*, and is often sought in conjunction with finding the global maximum. Assumptions such as D_0 compact and f continuous are usually placed on the problem, guaranteeing the existence of a global maximum and at least one global maximizer.

In practice the problem is relaxed somewhat to: given $\epsilon \geq 0$, find a function value y^* such that $y^* \geq f(\mathbf{x}) - \epsilon$ for all $\mathbf{x} \in D_0$. Such a y^* is called a *global maximum to within* ϵ. A point $\mathbf{x}^*$ such that $f(\mathbf{x}^*)$ is a global maximum to within ϵ is called a *global maximizer to within* ϵ. A further relaxation requires finding a global maximum to within ϵ with probability greater than some specified value.

The (relaxed) global optimization problem is usually denoted as

$$\max_{\mathbf{x} \in D_0} f(\mathbf{x}).$$

The problem of finding the global minimum (to within ϵ) is an equivalent problem since $\min\{f(\mathbf{x}) : \mathbf{x} \in D_0\} = -\max\{-f(\mathbf{x}) : \mathbf{x} \in D_0\}$. Either problem is called global optimization.

Throughout this thesis y^* is used to denote both the global optimum (minimum

or maximum) and a global optimum to within ϵ. It will be clear from the context which is meant. Similarly, $\mathbf{x}^*$ is used for both a global optimizer (minimizer or maximizer) and global optimizer to within ϵ and X^* for the set of all such points.

1.2 Local Optimization

The problem of local optimization is to find a point that is an optimizer of the function restricted to some neighbourhood of the point. This field is relatively well studied and there exist many standard algorithms for solving certain classes of functions efficiently. In general, algorithms for local optimization problems are not applicable to global optimization problems.

It is assumed that the reader is at least cursorily familiar with the theory and practice of local optimization.

1.3 Applications—Theoretical Versus Practical

The literature claims that many practical problems in engineering, physics, chemistry, economics, management, statistics and mathematics can be formulated as global optimization problems. The idea is that many practical problems can be modelled by an objective function that measures the "goodness", in some sense, of possible solutions to the problem. The global optimum of this objective function then represents the "best" solution to the problem, at least in terms of the model. However, after careful examination, it is found that while global optimization is useful for some theoretical problems and as a theoretical groundwork for other optimization methods, the global optimum is often not the best solution in a more holistic sense for many practical problems.

Careful inspection of practical problems reported in the literature reveals that while finding a local optimum is often not sufficient to solve the problem, since there may be many local optima, finding the (approximate) global optimum is not necessary since other solutions will be accepted. Most typically, finding a solution to the not so well-defined problems: "find a good solution fast" or "find a better solution than anyone else can find", is sufficient to solve the original practical

problem. In this thesis the term "non-local optimization" is used to describe optimization problems and algorithms for which local optimization is insufficient and global optimization unnecessary. (Unfortunately and rather confusingly, "non-local optimization" algorithms are often called "global optimization algorithms" by their authors.)

In order to examine the above claims let us consider four potential applications of global optimization. The first three of these are representative of practical problems: first, the central problem from computational chemistry, that of finding the structure of molecules; second, from economics, the well known plight of the travelling salesman who wishes to travel to numerous cities and back home with as little cost as possible; and third, from scientific, mathematical and statistical modelling, the problem of finding parameters for models that agree with experimental data to within some "experimental error". The fourth example is representative of a more theoretical problem, that of solving a hundred year old mathematical conjecture about minimizing surface areas that enclose volumes.

1.3.1 Molecular Conformations

One way of attempting to find the structure of a molecule is to study the potential energy of possible structures. For instance, the *Born-Oppenheimer surface* gives good approximations to the potential energy of a molecule in terms of its atomic positions [11]. This can be modelled, for instance, by considering forces associated with bond stretching, bending and twisting, and van der Waals forces between non-bonded atoms. More accurate approximations could, at least theoretically, be found by considering a full quantum mechanic model of the molecule.

The structure of a molecule is not static, but always changing due to thermal motions and quantum effects. Stable structures, or *conformations*, of a molecule are generally believed to correspond to local minimizers of its potential energy surface ("although there is no direct experimental or theoretical evidence for this [for protein molecules]" [10, page 9] and, in fact, it is not true for some molecules, such as IHI, where there is no finite energy minima and the observed structure is close to a saddle point [49]). Exceptions aside, it is often assumed that molecules in nature will be found in (or close to) their "most stable conformation", the conformation that corresponds to the global minimum potential energy. To quote Burkert and

Allinger [11], "[w]ith ... complicated molecules, there will in general be a large number of energy minima of different depths. To a first approximation, the molecule is described by the structure corresponding to the deepest energy minimum [a global minimizer]."

This is the premise on which many global optimization papers discussing molecular conformation problems have been based (see, for instance, [51, 52] special issues of the *Journal of Global Optimization* dedicated to this subject). However, while it may be true that molecules are more likely to be found in nearby lower energy conformations than higher energy conformations, this does not necessarily mean they are most likely to be found in a conformation corresponding to a *global* minimizer of the potential energy surface. Imagine, for instance, a long cyclic chain of bonded atoms that starts in a low energy conformation topologically equivalent to a knot which is not topologically equivalent to the global minimizer. While thermal motion, quantum tunnelling and other effects may move the molecule out of local minima to lower energy levels, it is very unlikely that the molecule will find the global minimum conformation in any reasonable time. Doing so would involve passing a very high energy barrier (such as the chain passing through itself).

Evidence against the claim that molecules are usually found in their global minimum energy conformation can be found in nature. For instance, Prusiner [55] has identified certain proteins, known as *prions*, which are normally found in nature in what appears to be only a local minimum conformation [49]. An apparently lower energy conformation of these molecules, called "scarpie" prions, act as a catalyst to convert "normal" prions to the "scarpie" conformation. Without this catalytic action the apparently lower energy conformation may never have been experimentally or computationally detected.

DNA molecules give another example. DNA molecules are found in nature in a well-defined unique conformation that resembles a left-handed helix wound up into a ball. The fact that these molecules are never found in the optical isomer of this conformation, a right-handed helix wound up into a mirror reflection of the ball, suggests that the natural conformation depends on more than just potential energy. If molecular structures did depend only on their potential energy, it would be equally likely to find DNA molecules in either isomer. In fact, it may be the case that the natural conformation for DNA molecules is not even close to a global minimizer as

is often presumed.

It is common practice in molecular mechanics to start with a conformation which is thought to be close to the actual conformation (found by, for instance, X-ray crystallography) and to "minimize" from this starting point. Such a minimization can return poor predictions if the method is purely local since it can get stuck in "small basins" far from the actual conformation. However, other "non-local optimization" algorithms, such as simulated annealing, can jump out of these "small basins" to find nearby conformations of lower potential energy, which may turn out to be much better predictors of actual structures. It should be noted that while these "non-local optimization" algorithms are often called "global optimization algorithms" they are not guaranteed to solve global optimization problems in finite time.

In conclusion, the fact that some "non-local optimization" methods sometimes return good predictions of actual structures should not be used as evidence that molecules are found in their global minimum energy conformations. It may be that the successful "non-local optimization" algorithms merely find the same non-global low energy conformations as the molecules themselves. In terms of predicting actual molecular conformations, seeking the global minimum potential energy conformations may therefore be a red herring, and global optimization may not be useful. On the other hand, for theoretical problems, such as determining the theoretically most stable structures and settling the question of whether or not these are the natural structures for molecules, global optimization has the potential to be invaluable.

1.3.2 The Travelling Salesman

It is claimed in the literature that many economic problems can be solved by formulating the problem as an objective function and finding the global minimum. This is epitomised by the well-known plight of the travelling salesman. By career choice, the travelling salesman starts from his hometown, travelling to a number of other cities selling his wares, before returning home. By experience, he knows the cost of travelling between any two cities. By desire, he wishes to retire a rich man, or at least as rich as a travelling salesman can hope to be.

Perhaps the first advice for the travelling salesman was the 1832 German book *Der Handlungsreisende, wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiss zu sein. Von*

eigem alten Commis-Voyageur ('The Travelling Salesman, how he should be and what he should do to get Commissions and to be Successful in his Business. By a veteran Travelling Salesman')[31]. More recently, there have been thousands of papers written by mathematicians, computer scientists, management scientists and others with the mythical travelling salesman in mind (the INSPEC CD-ROM cites 1050 papers which refer to "traveling[sic] salesman" published between 1989-96).

The reader will surely agree with the general consensus that if the travelling salesman, by god-given revelation, inherited knowledge, commissioned research or any other method, *knew* the tour of least cost, he should use it. But what if he does not know?

The problem of finding the minimum cost tour, given the cost of travel between each pair of cities, is known as the *Travelling Salesman Problem (TSP)*. TSP is a global optimization problem and there are many algorithms that solve it. It is well-defined, elegant and leads to interesting and rich mathematics. The reader is referred to [38] for an excellent exposition of TSP.

Should the travelling salesman definitely solve, or commission someone to solve, the travelling salesman problem? No, because the travelling salesman problem does not take account of the cost of finding the tour of least cost. It is well known that TSP is very hard to solve. It may take many years and billions of dollars to find a solution. Such a cost is not conducive to a prosperous retirement.

For any economic problem, the cost of finding a solution should be taken into account. In general, global optimization does not take account of this cost and, unfortunately, how to include this cost into an objective function is not at all clear. There are many "non-local optimization" algorithms that attempt to solve the less well-defined problem "find a good solution fast" or "find a better solution than anyone else can find". In practice, these algorithms solve economic problems more efficiently than global optimization algorithms since, at least to some extent, the cost of finding the solution is included in a heuristic sense in the problem's formulation. On the other hand, for theoretical problems, such as finding the theoretical best solution and evaluating a "non-local optimization" algorithm's performance, global optimization is needed.

1.3.3 Mathematical Modelling

Often in the mathematical sciences one wishes to model or summarise data by finding a *fitting function* which approximately agrees with the data. For instance, fitting a statistical distribution to sampled data, modelling a physical process for which experimental data is known, or simplifying a complicated, but known, function.

There may be reason for believing the fitting function should have a certain form, but with unknown parameters. The problem is then to find the "best" set of those parameters to fit the function to the data. If we interpret "best" to be best in the least-squares sense, for instance, the problem can be formalised as

$$\max_{\theta} \left(-\sum_{i}^{k} (F(\theta, \mathbf{x}^i) - y^i)^2 \right)$$

where θ is the set of parameters of the fitting function F and $(\mathbf{x}^i, y^i)$, $i = 1, \ldots, k$ are the data. Here a global maximizer is sought in conjunction with the global maximum.

For some fitting functions, for instance, if F is linear, polynomial or a spline, this problem can be solved by standard methods. These methods are *tractable*, that is, they can be solved in a time that is bounded above by a polynomial of the problem size. However, in general, for non-linear fitting functions, the objective function described above may have many local optima that are far from the global optimum and the problem may be *intractable*.

There are standard algorithms for solving least-squares problems, notably in the NAG library of routines [50]. However, for general fitting functions these algorithms find only a local optimum, not necessarily the global optimum. In any case, if a set of parameters is found, by a standard algorithm or any other method, which give a fitting function of the data that is within the desired tolerance, the problem is solved.

If standard algorithms (or other methods) cannot find a suitable set of parameters then one is left with two possibilities. Firstly, the fitting function may be inappropriate—there may be no solution to the problem. In this case, a new fitting function should be tried and the process of attempting to find a suitable set of parameters repeated. Alternatively, there may be a solution that the standard algorithms do not find. This may be the case if there are strong theoretical reasons

for the choice of the fitting function. In this case, a good "non-local optimization" algorithm may be able to find a solution, where the standard algorithms fail. If even this fails to find a suitable set of parameters, then, finally, a global optimization algorithm might be appropriate to try to find a solution or prove that none exist.

Thus, for the practical problem of modelling data, the standard methods to find suitable sets of parameters (generally, local and "non-local" optimization algorithms) are preferred over global optimization algorithms. However, for some theoretical problems, an important example being proving a fitting function cannot fit the data no matter how the parameters are chosen, global optimization (or its equivalent) is necessary.

1.3.4 Mathematical Problems

There is, of course, an inexhaustible supply of mathematical global optimization problems. For instance, one interesting application is solving the hundred year old "double-bubble conjecture":

Conjecture 1.3.1 *The minimum area surface which encloses two regions of volumes v_1 and v_2 is a* double-bubble, *a surface made of three pieces of spheres meeting along a circle at an angle of* $120°$.

This conjecture was recently solved in the case of equal volumes, that is $v_1 = v_2$, by Hass, Hutchings and Schlafly [29]. Using ingenious geometric arguments, they first show that the minimum surface is either a double-bubble or a torus-bubble (see Figure 1.1). The problem can then be solved by showing that there is some $\epsilon \geq 0$

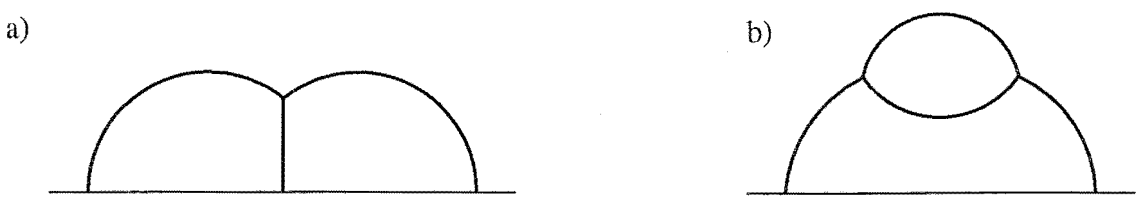

Figure 1.1: A double-bubble is the surface of revolution of (a). A torus-bubble is the surface of revolution of (b).

such that the global minimum surface area to within ϵ of any torus-bubble enclosing

equal volumes, which can be described by two real parameters, is greater than the surface area of the double-bubble. (This, in fact, is similar to what they did.)

For such mathematical problems, "non-local optimization" is not sufficient to prove the result. If and only if it is necessary to find the global optimum (or do equivalent computations) in order to solve the problem can the problem inarguably be called a global optimization problem.

1.3.5 Conclusions

There are very few practical problems for which the (approximate) global optimum of a suitably defined objective function is the only acceptable solution. In an attempt to find such problems the author appealed to the Internet community and academics—he found none. Often practical problems can be formulated and solved as "non-local optimization" problems.

On the other hand, there are some theoretical and mathematical problems for which finding the global optimum is the only way of solving the problem. For these problems, good global optimization algorithms may be invaluable.

For any optimization problem, one should use global optimization only if either finding the global optimum is essential to solving the problem no matter what the cost, or that the cost of finding the global optimum is negligible. The latter is unlikely for large classes of functions since the problem is known to be intractable (unless $NP = P$), although for small classes, such as linear functions on polytopes, global optimization is already the norm (since local optimization and global optimization are equivalent).

Finally, the field of global optimization leads to interesting and rich mathematics in its own right, which justifies its study.

1.4 A Survey of Algorithms

This section gives a brief survey of some global optimization algorithms from the literature. It covers the main types of global optimization algorithms that work on fairly general classes of functions, giving an indication of the diversity and richness of the field. Not covered are the many "non-local optimization" algorithms, which do not necessarily solve the global optimization problem (or have not been proven

to do so), or algorithms which work only on functions in a very specific class. The reader is referred to [68] for a good survey of many techniques and a very extensive bibliography of many global optimization algorithms and heuristics. Also, see [32] for a more recent publication containing papers on many areas of global optimization.

1.4.1 Pure Random Search

Perhaps the very simplest of ideas in global optimization is Pure Random Search (PRS). The idea is simply to sample points chosen uniform randomly over the whole domain until sufficiently confident that the largest of these values is a global maximum to within ϵ.

In order to achieve such confidence, it is necessary to know the probability that each independent sample is a global maximum to within ϵ. This probability p_1 is equal to the relative volume (or measure) of X^* compared to the volume of D_0. Then, $\lceil \ln(1-p)/\ln(1-p_1) \rceil$ sample points are required to find the global maximum to within ϵ with probability at least p.

A modification of this idea is *multi-start*. In multi-start, a number of starting points are chosen uniform randomly over the whole domain, and a local optimization algorithm is started from each of these points. *The region of convergence* is defined to be the set of points for which the local optimization algorithm converges to a global optimizer to within ϵ. Thus, if any of the starting points are in the region of convergence, multi-start will return the global optimum to within ϵ. Therefore, $\lceil \ln(1-p)/\ln(1-\alpha) \rceil$, where α is the relative volume of the region of convergence, starting points are required to find the global optimum to within ϵ with probability at least p. Generally, the volume of the region of convergence is greater than the volume of X^*.

In practice, it is very unusual that the parameters p_1 or α, or even lower bounds on these, are known. Also, for high dimensional functions, these parameters are likely to be very small. For instance, suppose $f(\mathbf{x})$ is defined on the unit ball in $\mathbb{R}^n$, and the region of convergence is a ball of radius $1/2$. To have 95% confidence of finding the global minimum to within ϵ, only 5 starting points are required if $n = 1$, but more than 3.7×10^{30} starting points are required if $n = 100$.

1.4.2 Pure Adaptive Search

Pure Adaptive Search (PAS) [73], while not likely to be a practical algorithm, is interesting as a theoretical approach due to its excellent convergence results. The *level set of* y is defined as the set of points $\mathbf{x} \in D_0$ such that $f(\mathbf{x}) \geq y$. In the kth iteration, PAS samples a point $\mathbf{x}_k$ chosen uniform randomly over the level set of the highest known function value. If the level set of $f(\mathbf{x}_k) + \epsilon$ is empty, then a global maximum to within ϵ is known. With mild assumptions, the expected number of function evaluations required by PAS to find the global optimum to within ϵ is $O(n \log(1/\epsilon))$, where n is the dimension of the problem.

It is necessary to know all the level sets of f in order to implement PAS efficiently, and to be able to choose a point uniform randomly over these sets. Except for very special functions this is not likely to be efficiently realisable in practice.

It is sometimes thought that it is the stochastic nature of PAS that leads to its excellent convergence. However, it is possible to have theoretical deterministic algorithms with comparable rates of convergence and requirements. For instance, suppose that the level set of u is empty and the level set of l is not. Then, if the level set of $(u+l)/2$ is empty, we know $y^* < (u+l)/2$. If not, then $y^* \geq (u+l)/2$. Thus, we can perform a binary search on the range of y^*, requiring only $O(\log((u-l)/\epsilon))$ steps to find the global optimum to within ϵ.

1.4.3 Lipschitz Algorithms

Piyavskii [54], and independently Shubert [62], discovered an algorithm for univariate Lipschitz continuous functions defined on an interval. In general, multivariate Lipschitz continuous functions are defined as follows.

Definition 1.4.1 *A function* $f : D_0 \to \mathbb{R}$ *is called* Lipschitz continuous *if there exists* $L > 0$ *such that*

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq L\|\mathbf{x} - \mathbf{y}\|, \forall \mathbf{x}, \mathbf{y} \in D_0.$$

Such a constant L *is called a* Lipschitz constant.

The Piyavskii-Shubert algorithm has been directly extended to the multivariate case by Mladineo [42] and, using techniques similar to Breiman and Cutler's algorithm

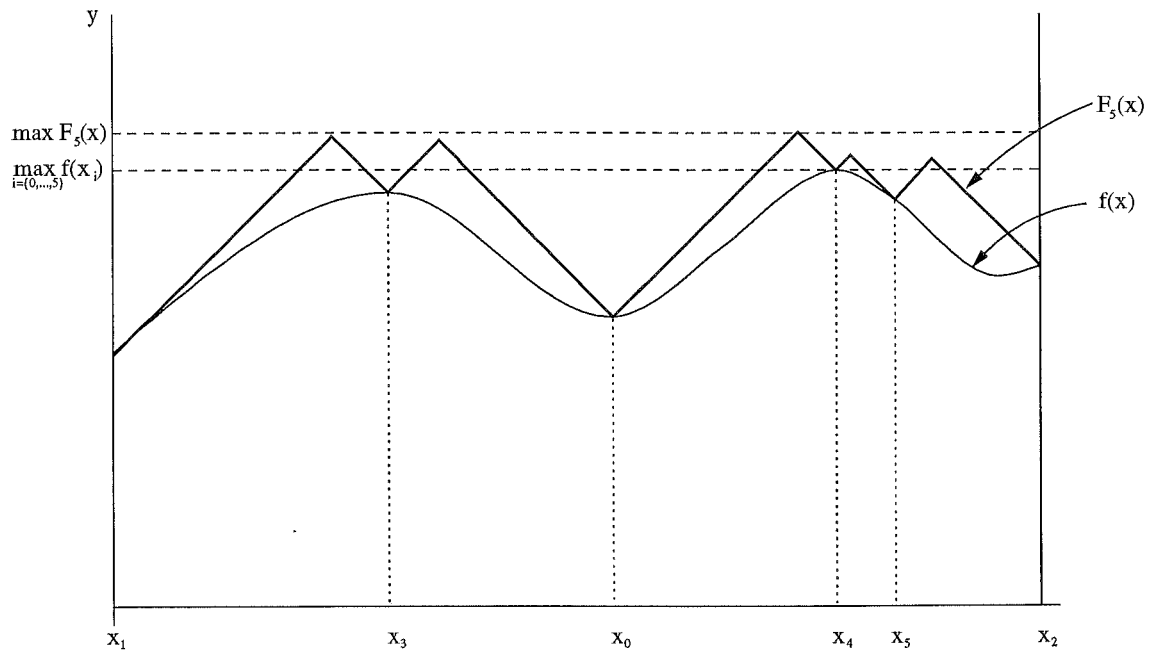

Figure 1.2: The Piyavskii-Shubert saw-tooth cover of $f(x)$ after six function evaluations.

discussed below, improved upon by Jaumard, Herrmann and Ribault [26]. The reader is referred to this last reference for a comprehensive exposition of Lipschitz algorithms.

The basis of the method is that after evaluating at k points in D_0, $\{\mathbf{x}_0, \ldots, \mathbf{x}_k\}$, a piecewise continuous function

$$F_k(\mathbf{x}) = \min_{i=0,\ldots,k} \left(f(\mathbf{x}_i) + L\|\mathbf{x} - \mathbf{x}_i\|\right)$$

can be defined. Thus, $F_k(\mathbf{x})$ is made up of cones whose vertices are at evaluation points on the graph of $f(\mathbf{x})$. The functions $F_k(\mathbf{x})$ have the property that, for all $\mathbf{x} \in D_0$, $F_k(\mathbf{x}) \geq f(\dot{\mathbf{x}})$. Because of their shape (see Figure 1.2) these functions are sometimes called *saw-tooth covers* in the univariate case. More generally they are called *upper envelopes of* f.

Definition 1.4.2 *A function* $F : D_0 \to \mathbb{R}$ *is called an* upper envelope of f *if*

$$F(\mathbf{x}) \geq f(\mathbf{x}), \ \forall \mathbf{x} \in D_0.$$

The Piyavskii-Shubert algorithm evaluates the objective function initially at the midpoint of the interval, and thereafter at a maximizer of $F_k(x)$. The algorithm

stops when the *variation* at the kth iteration,

$$\max_{x\in[a,b]} F_k(x) \; - \max_{i=0,\ldots,k} f(x_i),$$

is not greater than ϵ. It halts after a finite time for any $\epsilon > 0$ with the greatest function evaluation being a global maximizer to within ϵ.

Mladineo's algorithm is similar, but finding a maximizer of $F_k(\mathbf{x})$ is harder. Jaumard, Herrmann and Ribault's algorithm make use of the underlying network of intersections of the cones to ease this task (see Figure 1.3). This network can be updated relatively efficiently after each function evaluation. Also, Wood's algorithm [72] generalises Piyavskii-Shubert's algorithm to the multivariate case but, rather than cones, uses simplexes to make up the upper envelope. This simplifies the finding of maximizers of $F_k(\mathbf{x})$ considerably.

It is often quoted in the literature that the Piyavskii-Shubert algorithm is a *one-step minimax optimal* algorithm. That is, the maximum variation, over all possible objective functions, is minimized at each iteration. The following remark demonstrates this is not strictly the case.

Remark 1.4.1 *Suppose the objective function $f : [0,1] \to \mathbb{R}$ has Lipschitz constant one and $f(1/2) = 1/2$ and $f(0) = 0$ are the first two function evaluations. The Piyavskii-Shubert algorithm's next evaluation is at $x_2 = 1$ which leads to a worst case variation of $1/4$, achieved when the function value is equal to 1/2. Evaluating instead at $x_2' = 5/6$ gives a worst case variation of $1/6$ achieved when the function value is greater than or equal to 1/2 (cf. [35]). See Figure 1.4.*

For the univariate case this is a minor point, since the Piyavskii-Shubert does become one-step minimax optimal after the first three function evaluations, which are always at the mid-point and the end-points of the interval. Of more importance, Mladineo makes the same claim for her multivariate extension of Piyavskii-Shubert assuming incorrectly that the maximizer of the upper-envelope is always at the intersection of $n+1$ cones. In fact the maximizer may be at the intersection of $m < n+1$ cones and $n+1-m$ boundary hyperplanes. Since there are an infinite number of boundary points for $n > 1$, the algorithm may never revert to a one-step minimax optimal algorithm.

Minimax optimality can be viewed as a pessimistic optimality criterion—the variation is minimized in the worst case. A more optimistic one-step optimality

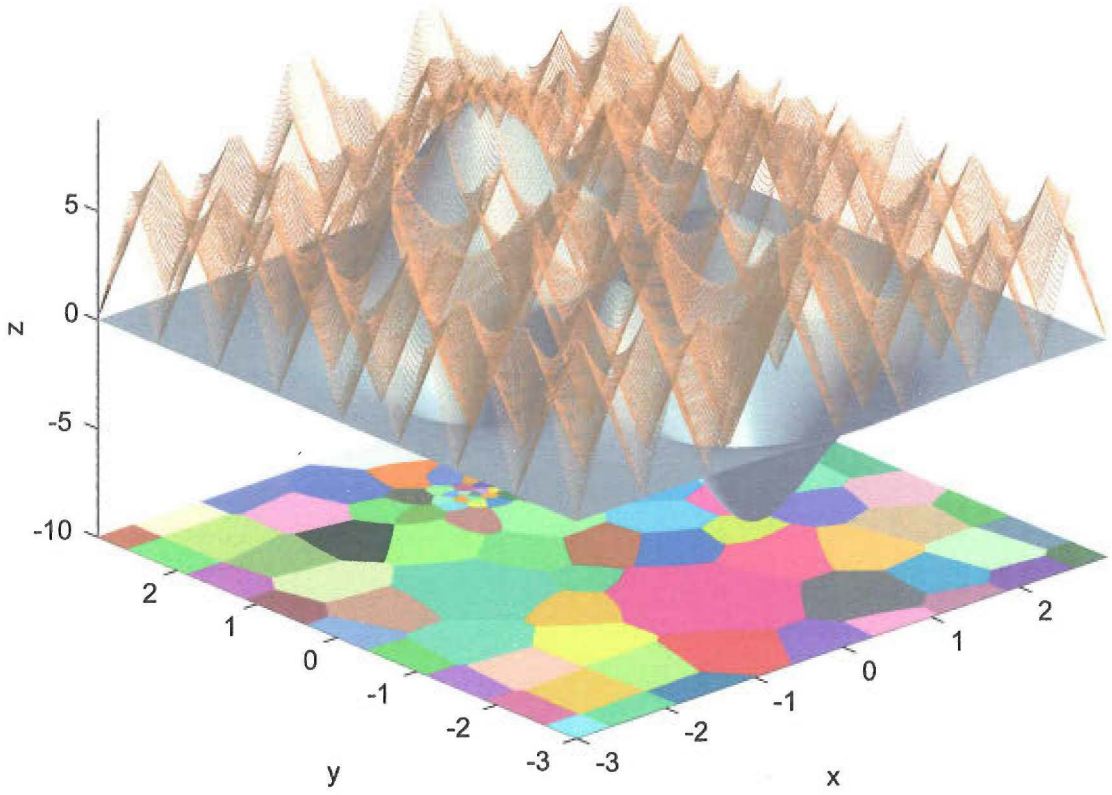

Figure 1.3: The Mladineo upper envelope (copper wire-frame) with $L = 15$ on the Peaks functions (grey surface) after 100 iterations. Underneath are the projections onto the domain of the cones that make up the upper envelope (randomly coloured). The curvilinear edges and vertices of these projections form a network used by Jaumard, Herrmann and Ribault's algorithm.

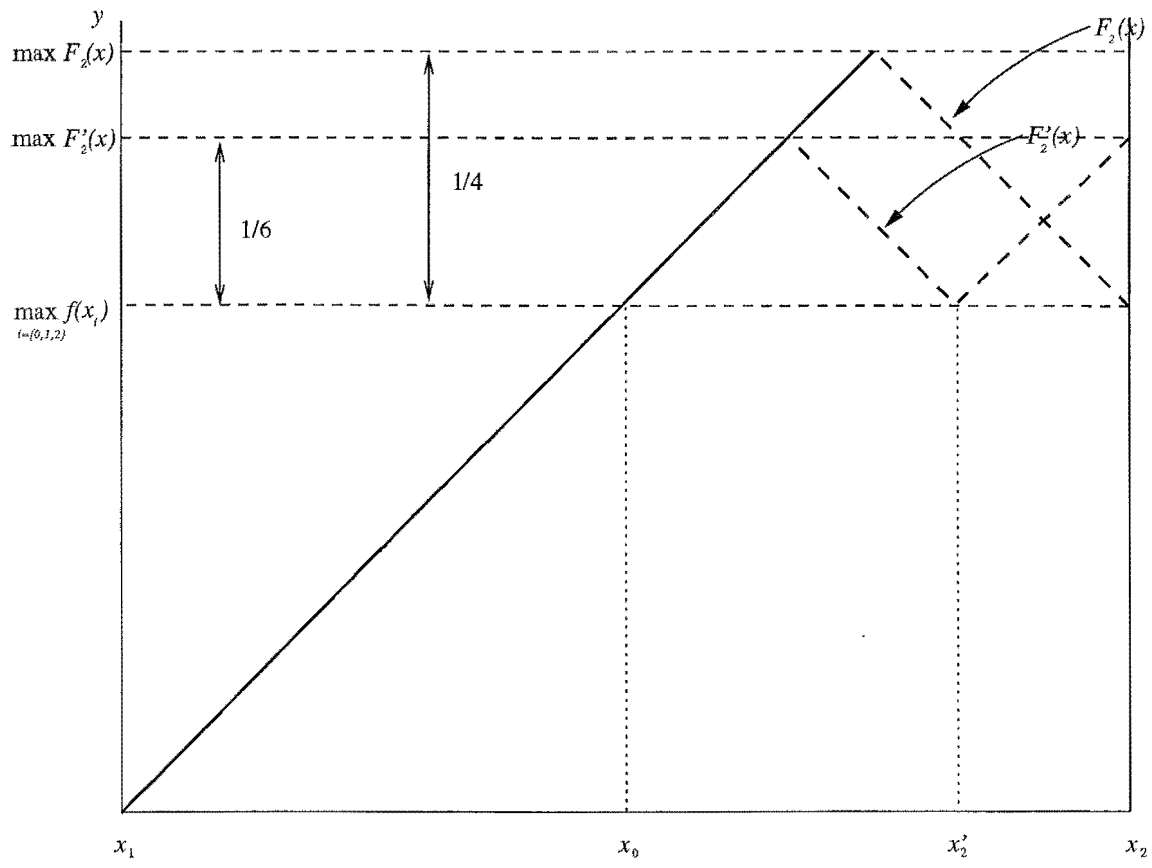

Figure 1.4: Counter example to minimax optimality of the Piyavskii-Shubert algorithm in the third iteration.

criterion, to minimize the minimum variation over all possible objective functions at each iteration, can be defined as follows.

Definition 1.4.3 *An algorithm is* one-step minimin optimal *on a class of functions $\mathcal{F}$ if, given $f \in \mathcal{F}$, for each iteration k there is at least one function $g_k \in \mathcal{F}$ satisfying $f(x_i) = g_k(x_i),\ i = 0, \ldots, k$ and the algorithm applied to g_k would halt with zero variation after one further function evaluation.*

It is easy to show that both Piyavskii-Shubert and Mladineo's algorithm are one-step optimal in this sense.

Proposition 1.4.1 *The Piyavskii-Shubert algorithm and Mladineo's algorithm are one step minimin optimal on the class of Lipschitz continuous functions with Lipschitz constant L after the first function evaluation.*

Proof: For any $k > 0$, let $g_k = F_k$. With one further function evaluation, the variation will be zero and the algorithm will halt. □

It is not clear whether either criterion has any rational relevance to the overall performance of an algorithm. (Schoen [60] defines an n-step optimality that may have more relevance to the overall performance of an algorithm.)

1.4.4 Second Derivative Algorithms

Using a bound on the second derivative, upper envelopes can be generated in a similar fashion to Lipschitz algorithms. This was first proposed by Brent [9], for the univariate case, and Breiman and Cutler [8] for the multivariate case. The algorithms are generalised by Baritompa and Cutler [5, 6].

In its simplest form, if u is a constant such that

$$f(\mathbf{x}) \leq f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x} - \mathbf{a}) + u\|\mathbf{x} - \mathbf{a}\|/2,\ \forall \mathbf{x}, \mathbf{a} \in D_0, \tag{1.1}$$

then after evaluating at $\{\mathbf{x}_0, \ldots, \mathbf{x}_k\}$,

$$F(\mathbf{x}) = \min_{i=0,\ldots,k} \left(f(\mathbf{x}_i) + \nabla f(\mathbf{x}_i)^T(\mathbf{x} - \mathbf{x}_i) + u\|\mathbf{x} - \mathbf{x}_i\|/2 \right)$$

is an upper envelope of $f(\mathbf{x})$.

This upper envelope is a piecewise quadratic function, and has the property that each quadratic piece is defined on a polytope of $\mathbb{R}^n$ containing an evaluation point

(see Figure 5.1 in Chapter 5). This is exploited in Breiman and Cutler's algorithm and makes it possible to find the maximizers of $F_k(x)$ relatively efficiently. After k evaluations, the next evaluation is chosen to be such a maximizer.

Chapter 4 of this thesis gives a method to obtain certain bounds on the second derivative, required for Baritompa and Cutler's generalisation of this method. These bounds can also be used to find a constant u satisfying Inequality (1.1) above. Chapter 5 discusses second derivative methods further and presents an adaptive second derivative method based on these methods.

1.4.5 Interval Methods

Interval arithmetic [43] can be used to provide bounds on functions by means of interval operations. In its simplest form, interval operations, $+, -, \cdot$ and $/$, are defined so that if X and Y are intervals of the reals, then $x \in X$ and $y \in Y$ imply that $x * y \in X * Y$, where $* \in \{+, -, \cdot, /\}$. With these operations, it is possible to bound any rational function over any box.

Interval arithmetic is used in two main ways (although they are often combined) for global optimization. Firstly, Newton's algorithm for locating stationary points of a real function can be extended to an *interval Newton algorithm* (first introduced by Moore [43]). Such an algorithm can be used to find *all* the stationary points of a real valued function, and hence all the global maximizers (assuming they are in the interior). Such methods are beyond the scope of this thesis. The interested reader is referred to [48] for the state-of-the-art in interval Newton methods.

Secondly, interval arithmetic can be used in a branch and bound framework for global optimization. For instance, the Ichida-Fujii [34] algorithm subdivides the domain into boxes. On each of these boxes the function value at the midpoint provides a lower bound on the global maximum. Interval arithmetic is used to find an inclusion of all the function values in each box, providing an upper bound on the global maximum. Any box with an upper bound less than the greatest lower bound over all boxes can be discarded, as it cannot contain a global maximizer. The Ichida-Fujii algorithm then subdivides the box with the greatest upper bound and finds bounds on the new subboxes. This process is continued until sufficient accuracy is achieved.

Chapter 3 discusses interval arithmetic in more depth and generalises it.

1.4.6 Integral Algorithms

While the global optimizers cannot be characterised by purely local criteria like the zero derivative characterisation of local optimizers, they can be characterised as the limits of integrals.

Pinkus [53] shows that if the objective function f has a unique global maximizer and f is bounded, then

$$x_i^* = \lim_{\lambda \to \infty} \frac{\int_{D_0} x_i \exp(\lambda f(\mathbf{x}))d\mathbf{x}}{\int_{D_0} \exp(\lambda f(\mathbf{x}))d\mathbf{x}},$$

where x_i^* and x_i are the ith coordinates of $\mathbf{x}^*$ and $\mathbf{x}$ respectively.

Falk [15] shows that if f is non-negative and

$$L = \lim_{k \to \infty} \int_{D_0} f(\mathbf{x})^k d\mathbf{x},$$

then if $L = \infty$, $y^* > 1$ and if L is finite then $y^* \leq 1$.

Chew and Zheng [12] show that

$$M(y) = \frac{1}{\mu(D_0(y))} \int_{D_0(y)} f(\mathbf{x})d\mathbf{x},$$

where $\mu(D_0(y))$ is the volume of the level set of y, is not less than y if and only if $y > y^*$.

Algorithms based on these, and similar characterisations, are known as integral algorithms [14, 75, 12]. However, apart from exceptional cases, analytic solutions for the integrals are not obtainable. If numerical integration is used instead then the algorithms cannot be guaranteed to solve global optimization problems in finite time (see the results of Chapter 2).

1.4.7 D.C. Algorithms

A function $f(x) = g(x) - h(x)$, where g and h are convex functions, is known as a d.c. function (standing for difference of convex). A number of global optimization algorithms for d.c. objective functions (subject to d.c. constraints) have been proposed. The reader is referred to [33, 69] for details of the theory and algorithms since it is beyond the scope of this survey.

It is well known that any C^2 function can be expressed as a d.c. function. In Chapter 4 an explicit representation is given for programmable C^2 functions allowing such functions to be solved by d.c. algorithms.

1.4.8 Branch and Bound Algorithms

Many of the methods described above can be placed in a branch and bound framework. In this framework three rules need to be defined; a *choosing* rule, a *subdividing* rule and a *bounding* rule.

In the initial step the domain is subdivided into finitely many subdomains and bounds on the global maximum of the objective function restricted to the subdomain are found. Thereafter, one or more of the collection of subdomains is chosen, subdivided and bounded. These new subdomains replace the chosen subdomain in the collection. At any stage, any subdomain with an upper bound on the global maximum less than the greatest lower bound of the global maximum over all subdomains is said to be *fathomed* and can be deleted from the collection (as it cannot contain the global maximum). The algorithm is halted when the difference between the greatest upper and lower bounds on the global maximum over all subdomains is less than ϵ. Horst and Tuy [33] provide conditions under which Branch and Bound algorithms halt and return a global optimum to within ϵ.

Branch and Bound algorithms are discussed in more depth in Chapter 5 where an adaptive second derivative algorithm in the Branch and Bound framework is presented.

Chapter 2

Global Optimization Requires Global Information

The previous chapter introduced a number of global optimization algorithms. All of these algorithms require global information in the form of a parameter (e.g. relative measure on the set of global maximizers to within ϵ, relative measure of the region of convergence, a Lipschitz constant, a bound on the second derivative, the functional form), and they are guaranteed to find a global optimum to within ϵ with at least some given probability. One criticism of these algorithms is that global information is hard to obtain or simply may not be available.

Thus there is a desire to design algorithms which avoid the need for global information. A number of algorithms have been proposed with this in mind. For instance, the DIRECT algorithm of Jones et al. [35], Strongin's algorithm [67], algorithms of Gergel [61, pages 200–201] and Sergeyev [61], and simulated annealing [37, 3, 19] as used in practice. In this chapter it is shown that these algorithms, and indeed all algorithms which avoid global information, have inherent theoretical limitations.

Inherent limitations of algorithms which stop after a finite stage are well known. Solis and Wets [64] point out that "the search for a good stopping criterion seems doomed to fail", because as noted by Dixon "even with [the domain] compact convex and f twice differentiable, at each step of the algorithm there will remain an unsampled square region of nonzero measure v (volume) on which f can be redefined (by means of spline fits) so that the unsampled region now contains the global [optimum]." Thus, after the run, it is possible that the algorithm failed. The results

in this chapter strengthen this by implying the existence of functions, a priori, for which the probability of success is arbitrarily small.

Some limitations for algorithms which are allowed to run forever are also known. Hansen et al. [28] found a class of functions for which Strongin's algorithm fails to converge. It is well known that any deterministic algorithm which uses only function values at sample points converges to the global optimum on all continuous functions if and only if it searches a dense set. (Törn and Žilinskas [68] provide a proof for this).

This chapter extends the above results. It is shown that the result reported in Törn and Žilinskas extends to any deterministic algorithm which uses only local information. Such algorithms converge to the global optimum on all functions in any "reasonable" class if and only if they always search a dense set of the domain. Intuitively, algorithms which additionally have the sample points converging only to global optimizers cannot also search a dense set. It is shown, with only local information, such algorithms fail on any "reasonable" class.

Introducing a stochastic element to algorithms is often seen as a way to overcome these limitations, so that no function can be found that will definitely fail. However, it is shown that there are analogous results for stochastic algorithms. For instance, for simulated annealing as used in practice there are functions for which the probability of success is arbitrarily small.

2.1 Definitions and Notation

The results in this chapter require very few conditions on the domain of the objective function D_0. It is assumed to be a compact subset of R^n with no isolated points. Note, objective functions on a standard domain such as the closure of a bounded open subset, or a domain described by "reasonable" constraint functions (as is often the case), are included.

For the results in this chapter, the class of functions for which the algorithm is designed must contain sufficiently many functions. Intuitively, conditions are required that allow arbitrarily large modifications of a function on arbitrarily small neighbourhoods without affecting the function elsewhere.

Note that when we say a set is *open* we refer to the relative topology on D_0. The

following is a formal definition of the class of functions considered.

Definition 2.1.1 *A nonempty class of functions $\mathcal{F}$ is* sufficiently rich, *if it consists of continuous functions and for all $y \in \mathbb{R}$, $\mathbf{x} \in D_0$, $f \in \mathcal{F}$ and N open in D_0 containing $\mathbf{x}$, there exists $g \in \mathcal{F}$ such that $g(\mathbf{x}) = y$ and $g|_{D_0 \setminus N} = f|_{D_0 \setminus N}$.*

Commonly used examples are continuous functions, C^n, C^∞, continuous functions with a unique global optimum, Lipschitz continuous functions (with an arbitrary constant) and functions with Lipschitz continuous derivatives. Many non-standard classes of functions satisfy our definition. For example, continuous functions with continuous first partial derivative and multiple global optima.

The following is a formal definition of local information. Let $\mathcal{X}_{finite}$ be the set of all finite sequences in D_0.

Definition 2.1.2 Local information *for a family $\mathcal{F}$ is a function, LI, defined on $\mathcal{F} \times \mathcal{X}_{finite}$ such that for all $f, g \in \mathcal{F}$, $X \in \mathcal{X}_{finite}$ and N open in D_0 containing X, if $f|_N = g|_N$ then $LI(f, X) = LI(g, X)$.*

The range of a local information function is intentionally left unspecified as there are many diverse examples. Local information includes any information depending on function values and any "limit information" at a finite number of sample points. Examples of such "limit information" include (partial and directional) derivatives, and less common "limit information" such as the limiting fractal dimension at a point. Also, any formula depending on these examples (indeed, on any local information) is itself local information. Thus, the maximum sample point, the maximum slope between sample points and the interpolating polynomial through sample points are local information.

Examples of non-local information are the Lipschitz constant, the dependence of δ on ϵ (using the usual notation) for uniform continuity, bounds on higher derivatives, the level set associated with a function value, the measure of the region of attraction, the "depth" of the function, the number of local optima, the functional form and the global optimum itself. If the family of functions has an associated probability distribution, the induced probability distribution for any of the above examples is also non-local information.

In this thesis non-local information is called *global information*, since the results are concerned with showing that this is a necessary condition for guaranteeing convergence. Note, some non-local information, such as the Lipschitz constant of the function on a proper subset of the domain (sometimes called a local Lipschitz constant), is not global in the sense of being information about the function over the whole domain. Such information may still not guarantee convergence of an algorithm that uses it. More refined versions of the results could characterise which types of global information are sufficient.

The following formally defines the algorithms under consideration. We consider *sequential sampling algorithms* which "run forever" producing an infinite sample sequence when run on a function f, $X_f = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$. The partial sequence $(\mathbf{x}_0, \ldots, \mathbf{x}_k)$ is denoted X_k. The closure of a sequence X is denoted by $\bar{X}$ and the subsequential limit points by X'. Note X'_f is never empty, as D_0 is compact, and that $\bar{X}_f = X_f \cup X'_f$.

In general, the next sample point in a sequential sampling algorithm can depend on local and global information about the function and, for stochastic sequential sampling algorithms, an instance of a random variable. The results of this chapter apply to sequential sampling algorithms which use only local information, defined below.

Definition 2.1.3 *A* deterministic sequential sampling algorithm using only local information *on a class of functions $\mathcal{F}$ is a sequential sampling algorithm for which there is a local information function LI such that for all $f \in \mathcal{F}$, when running on f, $\mathbf{x}_{k+1}$ depends only on $LI(f, X_k)$.*

For instance, any algorithm for which the next sample point depends only on the function and derivative values of previous sample points is a deterministic sequential sampling algorithm using only local information. Note, in deterministic sequential sampling algorithms using only local information, the next sample point $\mathbf{x}_{k+1}$ is itself local information.

Definition 2.1.4 *A* stochastic sequential sampling algorithm using only local information *on a class of functions $\mathcal{F}$ is a sequential sampling algorithm for which there is a local information function LI such that for all $f \in \mathcal{F}$, when running on f, $\mathbf{x}_{k+1}$ depends only on $LI(f, X_k)$ and ω_{k+1}, an instance of a random variable.*

This definition allows algorithms to have some randomness in choosing the next sample point. Note that for stochastic sequential sampling algorithms X_f, X'_f and $\bar{X}_f$ are random variables that depend on $\omega = (\omega_1, \omega_2, \ldots)$. Denote an instance of X_f by $X_f(\omega)$.

Often algorithms in the literature are justified by showing that they "converge" in the limit. Using the above ideas, this is now defined formally. Since more than one objective function will be referred to in the proofs of the main results, denote the set of global optimizers of f by X_f^*. (With little change to the proofs X_f^* could instead be considered to be the set of global optimizers to within ϵ).

Definition 2.1.5 *A sequential sampling algorithm is said to* see the global optimum of f *if* $\bar{X}_f \cap X_f^* \neq \emptyset$.

That is, an algorithm sees the global optimum if and only if it samples a global optimizer at some stage or in the limit. This form of convergence is typical of algorithms for which the emphasis is on finding the global optimal value. For instance, the DIRECT algorithm and PRS use this type of convergence. Note that, for stochastic sequential sampling algorithms, "seeing the global optimum if f" is a random variable that depends on ω.

Definition 2.1.6 *A sequential sampling algorithm is said to* localise the global optimizers of f *if* $X'_f = X_f^*$ *(or weaker* $\emptyset \neq X'_f \subseteq X_f^*$*).*

That is, an algorithm localises the global optimizers if and only if the sample points converge only to global optimizers. Hence localising implies seeing. This form of convergence is more typical of algorithms which emphasise finding the location of (some of) the global optimizers in the limit. For instance, Sergeyev's algorithm and simulated annealing use this type of convergence. Again, for stochastic sequential sampling algorithms, "localising the global optimum of f" is a random variable that depends on ω.

2.2 Main Results

Both definitions of convergence might seem easy to satisfy since the considered algorithms run forever. However, the results show algorithms that do not use global information have inherent limitations.

Practical realisations of algorithms terminate with only an initial finite segment of the sample sequence produced, ideally with some indication of error. Clearly, this compounds the limitations (see, for instance, the example in Section 2.3).

2.2.1 Deterministic Case

The following are the main results for the case of deterministic algorithms. These theorems are (almost) special cases of the stochastic results. Since, in this case, X_f is not a random variable, but a determined sequence, the proofs are simple and quite intuitive.

Theorem 2.2.1 *Any deterministic sequential sampling algorithm using only local information on a sufficiently rich class of functions $\mathcal{F}$ sees the global optimum of every function $g \in \mathcal{F}$ if and only if $\bar{X}_f = D_0$ for every function $f \in \mathcal{F}$.*

Proof: It follows immediately from the definitions that sampling a dense set implies seeing the global optimum (of the same function in fact). For the converse, suppose that there exists a function $f \in \mathcal{F}$ such that $\bar{X}_f \neq D_0$. Since $\bar{X}_f \neq D_0$, there exists $\mathbf{x}^0 \in D_0 \setminus \bar{X}_f$. Take a neighbourhood about this point whose closure is disjoint from $\bar{X}_f$. Find another function g agreeing with the original function outside this neighbourhood and taking a value larger than the global maximum of f at $\mathbf{x}^0$. Since

$$LI(f, X_k) = LI(g, X_k), \ \forall k,$$

running the algorithm on g gives sample sequence $X_g = X_f$ and so fails to see the global optimum of g. □

Since localising implies seeing, it follows immediately that an algorithm localising for every g implies $\bar{X}_f = D_0$ for every f. However, the following is a stronger result.

Theorem 2.2.2 *For any deterministic sequential sampling algorithm using only local information on a sufficiently rich class of functions $\mathcal{F}$, there exists a function in $\mathcal{F}$ for which the algorithm fails to localise the global optimizers.*

Proof: Let f be a function for which $D_0 \setminus X_f^*$ is uncountable, (the existence of such a function is assured by the conditions on $\mathcal{F}$ and D_0 ([30, Theorem 2-80, page

88]). Suppose that $X'_f \subseteq X^*_f$. It follows that $\bar{X}_f \setminus X^*_f = X_f \setminus X^*_f$ which contains at most countably many points. As $D_0 \setminus X^*_f$ is uncountable, $\bar{X}_f \neq D_0$. The proof of Theorem 2.2.1 gives a function $g \in \mathcal{F}$ such that $\bar{X}_g \cap X^*_g = \emptyset$ so $X'_g \not\subseteq X^*_g$. □

2.2.2 Stochastic Case

The stochastic results presented in this section are analogous to the deterministic results of the previous section. Theorem 2.2.3, analogous to Theorem 2.2.1, shows that stochastic sequential sampling algorithms using only local information will see the global optimum frequently on all functions if and only if all points in the domain are frequently seen. Theorem 2.2.4, analogous to Theorem 2.2.2, shows that there always exist functions for which the probability of such algorithms localising the global optimizers is arbitrarily small.

The essence of the proofs in the stochastic case relies upon the same ideas as the deterministic case. However, quite a few technical difficulties had to be overcome because X_f is a random variable.

For a deterministic sequence, if $\mathbf{x}^0$ is not a limit point of the sequence then there exists a neighbourhood of points which are disjoint from the sequence's closure. The following lemma is a stochastic analogue to this, showing that a bound on the probability of a point being a limit point of a random sequence implies a bound on any point in a neighbourhood being in the sequence's closure.

Lemma 2.2.1 *Let X be any random sequence of points in D_0. If for some $\mathbf{x}^0 \in D_0$ and probability p,*

$$P\left(\mathbf{x}^0 \in X'\right) < p,$$

then there exists $\mathbf{y}^0 \in D_0$ and a neighbourhood N of $\mathbf{y}^0$ in D_0, such that

$$P\left(\bar{X} \cap \bar{N} \neq \emptyset\right) < p.$$

Furthermore, if $\bar{M}$ is any closed subset of D_0 and $\mathbf{x}^0 \notin \bar{M}$ then it is possible to choose $\mathbf{y}^0$ and N such that $\bar{N}$ and $\bar{M}$ are disjoint.

Proof: Consider the non-negative real valued random variable

$$R = \inf\{\|\mathbf{x} - \mathbf{x}^0\| : \mathbf{x} \in X, \mathbf{x} \neq \mathbf{x}^0\}.$$

Note, whenever X is constantly $\mathbf{x}^0$, $R = 0$. Clearly, $R = 0$ implies $\mathbf{x}^0 \in X'$ so $P(R = 0) \leq P(\mathbf{x}^0 \in X') < p$. By right-hand continuity of the cumulative distribution function, there exists $\delta > 0$ such that $P(R \leq \delta) < p$.

Therefore,

$$P\left(\bar{X} \cap B \neq \emptyset\right) \leq P\left(R \leq \delta\right) < p$$

where $B = \{\mathbf{x} : \|\mathbf{x} - \mathbf{x}^0\| < \delta, \mathbf{x} \neq \mathbf{x}^0\}$ is the punctured open ball of radius δ centred at $\mathbf{x}^0$.

Finally, since D_0 has no isolated points and $\mathbf{x}^0 \notin M$ there exists $\mathbf{y}^0 \in D_0 \cap B \setminus M$. Let N be a neighbourhood of $\mathbf{y}^0$ whose closure is contained within $B \setminus M$. Then

$$P\left(\bar{X} \cap \bar{N} \neq \emptyset\right) \leq P\left(\bar{X} \cap B \neq \emptyset\right) < p.$$

and furthermore, $\bar{N}$ is disjoint from M. □

We now give the stochastic analogue for Theorem 2.2.1.

Theorem 2.2.3 *For any probability p and any stochastic sequential sampling algorithm using only local information,*

$$P(\textit{algorithm sees the global optimum of } g) \geq p, \ \forall\ g \in \mathcal{F}$$

if and only if

$$P\left(\mathbf{x} \in \bar{X}_f\right) \geq p, \ \forall \mathbf{x} \in D_0, \ \forall f \in \mathcal{F}.$$

Proof: If the probability for each point in the domain being in $\bar{X}_f$ is greater than or equal to p, it follows immediately that the global optimum points (of the same function) have probability greater than or equal to p of being seen. For the converse, suppose that there exists $f \in \mathcal{F}$ and $\mathbf{x}^0 \in D_0$ such that $P(\mathbf{x}^0 \in \bar{X}_f) < p$. Clearly, $\mathbf{x}^0 \in X'_f$ implies $\mathbf{x}^0 \in \bar{X}_f$, so $P(\mathbf{x}^0 \in X'_f) < p$ and Lemma 2.2.1 gives a non-empty neighbourhood $N \subseteq D_0$ such that

$$P\left(\bar{X}_f \cap \bar{N} \neq \emptyset\right) < p.$$

Because $\mathcal{F}$ is sufficiently rich there exists $g \in \mathcal{F}$ such that

$$f|_{D_0 \setminus N} = g|_{D_0 \setminus N} \tag{2.1}$$

and

$$\max_{\mathbf{x} \in D_0} f(\mathbf{x}) < \max_{\mathbf{x} \in D_0} g(\mathbf{x}). \tag{2.2}$$

Since, $\forall k$, $\mathbf{x}_{k+1}$ depends only on $LI(f, X_k)$ and ω_{k+1}, it follows from (2.1) that if $\bar{X}_f(\omega) \cap \bar{N} = \emptyset$ then $X_f(\omega) = X_g(\omega)$ and so $\bar{X}_f(\omega) = \bar{X}_g(\omega)$. Therefore,

$$P\left(\bar{X}_g \cap \bar{N} \neq \emptyset\right) < p.$$

As the global optimizers of g are contained within N, P(algorithm sees the global optimum of g) $< p$. □

As in Section 2.2.1, it follows immediately that P(algorithm localises the global optimum of g) $\geq p$, for all $g \in \mathcal{F}$ implies $P(\mathbf{x} \in \bar{X}_f) \geq p$, for all $\mathbf{x} \in D_0$ and $f \in \mathcal{F}$.

In the deterministic case, Theorem 2.2.2 showed that attempts to localise are guaranteed to fail. For stochastic algorithms the existence of functions with *zero* probability of localising cannot be guaranteed. However, a stronger result than above, analogous to Theorem 2.2.2, can be obtained.

Theorem 2.2.4 *For any stochastic sequential sampling algorithm using only local information and any $\epsilon > 0$ there exists a function $f \in \mathcal{F}$ such that*

$$P(\textit{algorithm localises the global optimum of } f) < \epsilon.$$

Proof: Suppose, to the contrary, that for all $f \in \mathcal{F}$ and for some fixed $\epsilon > 0$,

$$P\left(X'_f \subseteq X^*_f\right) \geq \epsilon. \tag{2.3}$$

A contradiction is obtained by showing that this allows the construction of a function $g \in \mathcal{F}$ and a subset M such that $P(X'_g \subseteq M) > 1$!

Choose an integer $n > 2$ such that $1/n < \epsilon$ and let $\mathbf{x}^1, \ldots, \mathbf{x}^{n+2}$ be $n+2$ distinct points in D_0. The function g can be constructed in the following iterative manner.

For $i \in \{1, \ldots, n+2\}$, a function $f_i \in \mathcal{F}$ and open set N_i with $\bar{N}_i$ disjoint from $\{\mathbf{x}^{i+1}, \ldots, \mathbf{x}^{n+2}\}$ such that

$$P\left(X'_{f_i} \subseteq M_i\right) > i(n-1)/n^2 \tag{2.4}$$

where

$$M_i = \bigcup_{k=1}^{i} N_k$$

is found. The function $g = f_{n+2} \in \mathcal{F}$ and the subset $M = M_{n+2}$ give the desired contradiction.

The specific details are:

For $i = 1$, let N_1 be a neighbourhood of $\mathbf{x}^1$ whose closure is disjoint from $\{\mathbf{x}^2, \ldots, \mathbf{x}^{n+2}\}$. Since $\mathcal{F}$ is sufficiently rich there exists $f_1 \in \mathcal{F}$ such that $X^*_{f_1} \subseteq N_1$. Then from (2.3), (2.4) follows in the case of $i = 1$,

$$P\left(X'_{f_1} \subseteq N_1\right) \geq \epsilon > (n-1)/n^2.$$

For $i \leq n+1$, assume the required functions and sets exist. Since $\mathbf{x}^{i+1}$ is disjoint from M_i,

$$P\left(\mathbf{x}^{i+1} \in X'_{f_i} | X'_{f_i} \subseteq M_i\right) = 0 < 1/(ni-i).$$

By Lemma 2.2.1, there exists $\mathbf{y}^{i+1} \in D_0$ and a neighbourhood N_{i+1} of $\mathbf{y}^{i+1}$, whose closure is disjoint from $\bar{M}_i \cup \{\mathbf{x}^{i+2}, \ldots, \mathbf{x}^{n+2}\}$, such that,

$$P\left(\bar{X}_{f_i} \cap \bar{N}_{i+1} \neq \emptyset | X'_{f_i} \subseteq M_i\right) < 1/(ni-i)$$

or

$$P\left(\bar{X}_{f_i} \cap \bar{N}_{i+1} = \emptyset | X'_{f_i} \subseteq M_i\right) > (ni-i-1)/(ni-i). \tag{2.5}$$

Since $\mathcal{F}$ is sufficiently rich, there exists $f_{i+1} \in \mathcal{F}$ such that

$$f_{i+1}|_{D_0 \setminus N_{i+1}} = f_i|_{D_0 \setminus N_{i+1}}$$

and

$$X^*_{f_{i+1}} \subseteq N_{i+1}.$$

As in the proof to Theorem 2.2.3, if $\bar{X}_{f_i}(\omega) \cap \bar{N}_{i+1} = \emptyset$ then $X'_{f_{i+1}}(\omega) = X'_{f_i}(\omega)$. So that

$$\begin{aligned} P\left(X'_{f_{i+1}} \subseteq M_i\right) &\geq P\left(X'_{f_i} \subseteq M_i \text{ and } \bar{X}_{f_i} \cap \bar{N}_{i+1} = \emptyset\right) \\ &= P\left(X'_{f_i} \subseteq M_i\right) \cdot P\left(\bar{X}_{f_i} \cap \bar{N}_{i+1} = \emptyset | X'_{f_i} \subseteq M_i\right) \end{aligned}$$

From (2.4) (for i) and (2.5), it follows that

$$P\left(X'_{f_{i+1}} \subseteq M_i\right) > (ni-i-1)/n^2. \tag{2.6}$$

Observe $[(X'_{f_{i+1}} \subseteq M_i)$ or $(X'_{f_{i+1}} \subseteq N_{i+1})]$ implies $X'_{f_{i+1}} \subseteq M_i \cup N_{i+1}$, the events $(X'_{f_{i+1}} \subseteq M_i)$ and $(X'_{f_{i+1}} \subseteq N_{i+1})$ are mutually exclusive (as N_{i+1} is disjoint from

M_i) and $X^*_{f_{i+1}} \subseteq N_{i+1}$. Therefore (2.3) and (2.6) give

$$\begin{aligned} P\left(X'_{f_{i+1}} \subseteq M_{i+1}\right) &= P\left(X'_{f_{i+1}} \subseteq M_i \cup N_{i+1}\right) \\ &\geq P\left(X'_{f_{i+1}} \subseteq M_i\right) + P\left(X'_{f_{i+1}} \subseteq N_{i+1}\right) \\ &\geq P\left(X'_{f_{i+1}} \subseteq M_i\right) + P\left(X'_{f_{i+1}} \subseteq X^*_{f_{i+1}}\right) \\ &> (ni - i - 1)/n^2 + 1/n \\ &= (i+1)(n-1)/n^2. \end{aligned}$$

Thus (2.4) holds for $i + 1$. □

2.3 Examples

The constructed function g on which the algorithm (likely) fails, may have quite high global constants associated with it, e.g. for Lipschitz continuous functions, g may have a high Lipschitz constant. This is precisely the main point. If the Lipschitz constant of the objective function was known (even probabilistically) and used as a parameter to the algorithm, the constructed function g could be ruled out. An alternative viewpoint is that knowing the Lipschitz constant restricts the class of functions to one which is not sufficiently rich.

If a sequential sampling algorithm using only local information is terminated according to some stopping rule, there is a function from any sufficiently rich class for which the probability of seeing the global optimum is arbitrarily small (or zero for the deterministic case). To see this, note that such an algorithm can be modified by replacing the "stop" command with a loop to repeatedly sample the best point seen. If the original algorithm sees the global optimum, the new algorithm, which is a sequential sampling algorithm using only local information, localises it. Theorem 2.2.4 applies.

Strictly speaking, an algorithm using "hidden" local information on which the next sample point depends is not a sequential sampling algorithm. For instance, PAS could be implemented, albeit rather inefficiently, by internally sampling uniform randomly over the whole domain until a point in the level set of the highest known function value is found. This next sample point, thus, depends not only on local information at the previous sample points, but also on "hidden" local information, the function values of the internal sample points. Of course, any algorithm

that depends on "hidden" local information corresponds to, and suffers the same limitations of, one that does not hide the local information (in this case PRS).

Global information may in fact be disguised. For example, in Sergeyev's paper [61] there are parameters r and ξ involved in the algorithm. Careful reading of a convergence result for this algorithm says for every function there is an r^* dependent on ξ, past which convergence is always guaranteed. However the proof shows $r\xi$ must exceed a multiple of the overall Lipschitz constant involved. So $r\xi$ is in fact a global constant for the given function. Similar comments hold for Gergel's algorithm.

Simulated annealing provides another example where global information is disguised. "Standard" simulated annealing localises if the cooling schedule is slow enough. Hajek ([21] shows (in the finite discrete setting) a necessary and sufficient condition on the cooling schedule depends on the *depth of the objective function* (the minimum decrease in function value necessary to reach the global maximum from any point in the domain). Clearly, this is a global parameter. In the continuous case, the results here show the cooling schedule must depend on global properties. So, attempts to find a suitable (or optimize an existing) cooling schedule by pre-sampling or adjusting the cooling schedule on the run using sample points are doomed to failure. Theorem 2.2.4 shows that there are always functions in sufficiently rich classes for which the probability of success of such a scheme is arbitrarily small.

Other algorithms do not have hidden global information. The DIRECT algorithm of Jones et al. is guaranteed to see the global optimum and uses only local information. This is a result of looking in a dense set of the domain. This is really where the local information *becomes* global. In practice, however, one stops the algorithm after a finite number of steps and, if no global information is used for the stopping rule, there will always exist a function for which probability of seeing the global optimum is arbitrarily small.

2.4 Extensions

Theorems 2.2.1 and 2.2.3 can be generalised in the "only if" direction to other definitions of convergence. The proofs require only that if the algorithm "converges",

by whatever definition, then the global optimum can be extracted from local information depending on the sample sequence. For instance, if the algorithm "sees the global optimum", the global optimum is simply the maximum function value over the sample points. Another definition of convergence is used for Wood's algorithm, which "brackets" X^* at each iteration and is proven to converge in the sense that the infinite intersection of the brackets is equal to X^*. Wood's algorithm requires global information to do this, but conceivably another algorithm using only local information could attempt to approximate this approach. Since the global optimum could be obtained by sampling a point in the infinite intersection of brackets if the algorithm converged, then such an algorithm would converge in this sense on all functions in a sufficiently rich class only if it sampled a dense set.

The results in this chapter assume the algorithms sample only from the "feasible" set. Often functions are defined on a larger domain but one is interested in the global optimum when constraints are satisfied. Algorithms such as relaxed dual and penalty methods use infeasible sample points. The results in this chapter can easily be extended to handle this by appropriate reformulation.

Finally, the results in this chapter apply to algorithms attempting to find global information other than the (approximate) global optimum. All that is necessary is the appropriate definition of sufficiently rich class. The class needs to contain functions agreeing closely to others but having different global information.

2.5 Conclusions

Limitations on specific types of algorithms have been noted before. In this chapter the results have been extended to a more general class of (finitely terminating and infinite) algorithms, both deterministic and stochastic, that might utilise function and derivative values (indeed, any type of limiting information) but not global information. All of these algorithms have theoretical limitations.

Theorem 2.2.3 means such algorithms will succeed frequently on all functions if and only if all points in the domain are frequently seen. That is, the algorithms must use brute force. Theorem 2.2.4 shows that attempts to localise the global optimizers on all functions with such algorithms are doomed to failure. In practice algorithms

must stop after a finite time, and there exist functions for which deterministic algorithms fail or stochastic algorithms fail with arbitrarily high probability. So, if no global information about the problem is utilised, the function may be one on which the algorithm (likely) fails. The user cannot have justified confidence in the results.

Nor can the user remedy these theoretical limitations by estimating global properties by finite sampling of the function. In order to guarantee solving all global optimization problems in a sufficiently rich class, global information is required.

Chapter 3

Generalised Interval Arithmetic

In the pessimistic light of the previous chapter, the reader might feel that the global optimization problem is insoluble. To find the global optimum we require global information; but as with any global information, to find this we require global information, and so on. If the objective function is a "black-box" function, for which nothing is known except function (and derivative) values at sample points, then this goes on without end. However, if the function's expression is known then we already have *all* the information about the function there is to be known. Obtaining useful global information from the function's expression may be difficult. Here interval arithmetic provides an answer—it allows useful information about the function to be extracted from the function's expression.

This chapter, first introduces interval arithmetic which leads naturally to inclusion functions defined on boxes. Interval arithmetic is then generalised to allow inclusion functions defined on non-box domains, and to give improved inclusion functions defined on boxes, by making use of the structure of the objective function's expression. Examples are included to demonstrate how generalised interval arithmetic is used and can lead to improved performance for global optimization algorithms.

3.1 Introduction to Interval Arithmetic

Interval arithmetic has its roots in error analysis as a technique to automatically bound errors in computations on digital computers. Interval arithmetic on digital

computers dates back to Turing's team working on the ACE (Automatic Computing Engine) at the National Physical Laboratory between 1946 and 1948 [71]. It was Moore, however, who brought the pieces of interval arithmetic together, extending its use to bounding all errors, including bounding real valued functions [43].

Interval arithmetic is a useful tool for reliable computing (including reliable global optimization). Reliable computing is necessary, for instance, for rigorous computer proofs involving real numbers and arithmetic. Interval arithmetic is also a useful tool for global optimization in its own right. In this case, it is usual that the global optimum is sought to within an error that is many times machine accuracy. It is therefore (arguably) reasonable to neglect the effects of computational errors (defiling the original purpose of interval arithmetic) and to assume exact computations in the sequel (including finding exact local optima using a local optimization algorithm).

Interval arithmetic is presented in the literature ([43, 2]) as an extension of real arithmetic to the set of real closed intervals $\mathbb{I} = \{[a,b]|a,b \in \mathbb{R}\}$. In this thesis, capital letters X, Y, etc. are used to denote intervals and bold capitals $\mathbf{X} = (X_1, \ldots, X_n)$, $\mathbf{Y} = (Y_1, \ldots, Y_n)$, etc. to denote interval vectors or *boxes* (the Cartesian product of intervals). If $[a,b]$ is an interval then $\min([a,b]) = a$ and $\max([a,b]) = b$.

Real arithmetic is extended to intervals as follows. If $* \in \{+, -, \cdot, /\}$ is a binary operation on the reals, then

$$X * Y = \{x * y | x \in X, y \in Y\}$$

is a binary operation on the intervals, except that X/Y is undefined if $0 \in Y$. These operations on intervals can be explicitly calculated with at most four real operations. Specifically,

$$\begin{aligned}
[a,b] + [c,d] &= [a+c, b+d], \\
[a,b] - [c,d] &= [a-d, b-c], \\
[a,b] \cdot [c,d] &= [\min\{ac, ad, bc, bd\}, \max\{ac, ad, bc, bd\}], \\
[a,b]/[c,d] &= [a,b] \cdot [1/d, 1/c], \text{ where } 0 \notin [c,d].
\end{aligned}$$

Similarly, *elementary functions* on the reals, for example the trigonometric functions, hyperbolic trigonometric functions and their inverses, the exponential function, logarithms, absolute values and taking integer powers can be extended to the

intervals (non-integer powers can be implemented as $X^Y = \exp(Y \cdot \log(X))$ for $\min(X) > 0$). For instance, if $r(x)$ is an unary elementary function on the reals then

$$r(X) = \left[\min_{x \in X} r(x), \max_{x \in X} r(x)\right]$$

defines an unary elementary function on the intervals. Binary or n-ary elementary functions could also be extended similarly. For all of the elementary functions described above, which we can take to be our elementary functions, explicit calculations for the end points can be found with only a few (real) operations by making use of known properties.

Clearly, degenerate intervals $[a, a]$, where $a \in \mathbb{R}$, are homeomorphic to the reals and can be written as just a. Also, note that in the context of interval arithmetic, one must take care to distinguish between a function and a function's expression. In this chapter, if f denotes a function then, when the independent variable $\mathbf{x}$ is unspecified, $f(\mathbf{x})$ is used to denote the expression of f. When $\mathbf{x}$ is specified, $f(\mathbf{x})$ denotes the function value at $\mathbf{x}$ as usual.

A *programmable function* is a function with an expression that can be written down using only the arithmetic operators, precedence parentheses, elementary functions, constants and the independent variables. For instance,

$$f(\mathbf{x}) = \sin(x_1)(x_1 + x_2) + \cos(x_2)(x_1 - x_2) \tag{3.1}$$

is an expression of a programmable function. On the other hand,

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } x_1 \text{ is irrational} \\ 0 & \text{if } x_1 \text{ is rational} \end{cases}$$

is an expression of a non-programmable function. A formal recursive definition follows.

Definition 3.1.1 *If $f(\mathbf{x}) = C$, where $C \in \mathbb{R}$, or $f(\mathbf{x}) = x_i$, for any $i \in \{1, \ldots, n\}$, then $f(\mathbf{x})$ is an* expression of a programmable function. *Also, if $f(\mathbf{x})$ and $g(\mathbf{x})$ are expressions of programmable functions then $(f(\mathbf{x}) * g(\mathbf{x}))$, where $* \in \{+, -, \cdot, /\}$ and $r(f(\mathbf{x}))$, where r is an elementary function, are expressions of programmable functions.*

Unnecessary precedence parentheses and multiplication dots can be dropped. The domain of the function is taken to be all the points for which the expression is defined.

An expression of a programmable function can be written as a tree with the independent variables and constants at the leaves and binary operations and elementary functions at the nodes. Any expression corresponding to a subtree is called a *subexpression* (see Figure 3.1).

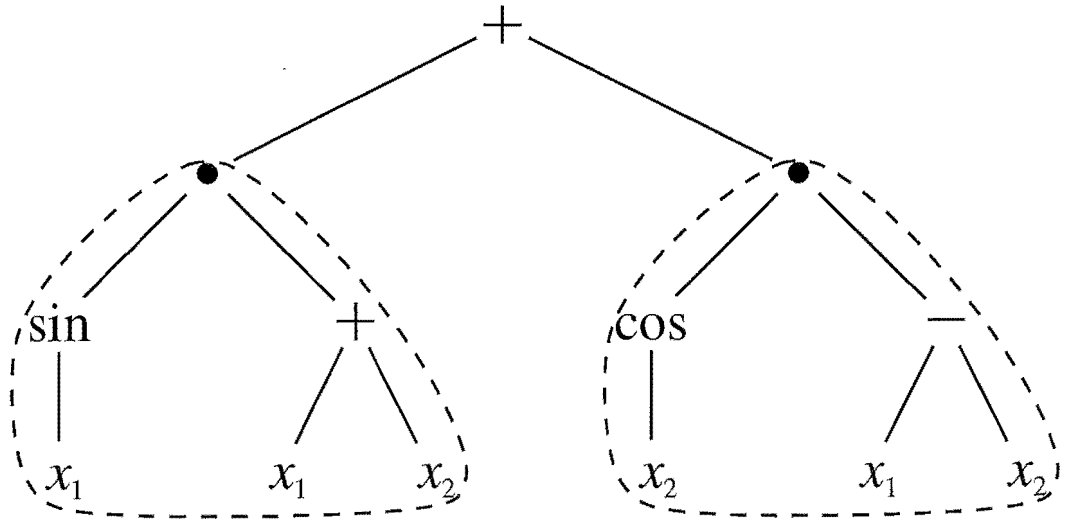

Figure 3.1: The expression tree for $f(\mathbf{x}) = \sin(x_1)(x_1 + x_2) + \cos(x_2)(x_1 - x_2)$. Any expression corresponding to a subtree, like the two marked, is called a subexpression.

The importance of interval arithmetic to global optimization (and to other fields to which it is applied) is that it provides a way to find "interval inclusion functions" for programmable functions.

Definition 3.1.2 *An interval function $F : \mathbb{I}^n \to \mathbb{I}$ is an* interval inclusion function *of a real function $f : \mathbb{R}^n \to \mathbb{R}$ if $\mathbf{x} \in \mathbf{X}$, where $\mathbf{X}$ is a box, implies $f(\mathbf{x}) \in F(\mathbf{X})$.*

The simplest interval inclusion function of a programmable function is the "natural inclusion function", defined below.

Definition 3.1.3 *Let $f(\mathbf{x})$ be an expression of a programmable function. The expression $f(\mathbf{X})$, found by replacing the real variables x_i in $f(\mathbf{x})$ with interval variables X_i for $i \in \{1, \ldots, n\}$, and treating the real constants, arithmetic operators and elementary functions as their interval counterparts, is the expression for the* natural interval inclusion function of $f(\mathbf{x})$.

For instance, the natural inclusion function of Expression (3.1) above is given by

$$f(\mathbf{X}) = \sin(X_1) \cdot (X_1 + X_2) + \cos(X_2) \cdot (X_1 - X_2).$$

The *fundamental property of interval arithmetic* states that the natural inclusion function is an interval inclusion function. So, for instance, an inclusion of $f : \mathbb{R}^2 \to \mathbb{R}$ defined by Expression (3.1) over the box $[0, 2] \times [-1, 4]$ is given by

$$\begin{aligned} f([0,2],[-1,4]) &= \sin([0,2]) \cdot ([0,2] + [-1,4]) + \cos([-1,4]) \cdot ([0,2] - [-1,4]) \\ &= [0,1] \cdot [-1,6] + [-1,1] \cdot [-4,3] \\ &= [-1,6] + [-4,4] = [-5,10]. \end{aligned}$$

Note that the tightest inclusion to one decimal place of f over this box is $[-0.6, 7.2] \subseteq [-5, 10]$.

Another important property of natural inclusion functions is the so called *inclusion monotonicity property* which states that $\mathbf{X} \subseteq \mathbf{Y}$ implies $f(\mathbf{X}) \subseteq f(\mathbf{Y})$ as long as $f(\mathbf{Y})$ is defined.

It is important to note that, while two different expressions may define the same real functions, the natural inclusion functions of these expressions can lead to different interval functions. For instance, the identity function can be written $f(x) = x$ or $f(x) = 2x - x$. But, the natural inclusion functions for these expressions are not equal since $[0, 1] \neq 2[0, 1] - [0, 1] = [-1, 2]$.

The larger inclusion of the second expression above is due to the so called *dependency problem.* Variables that occur more than once in the expression of a real function are dependent, but in an expression of an interval function they are treated as equal, but not dependent. Sometimes rewriting the function's expression can lead to tighter inclusions by reducing dependency. Also, Hansen [23] presents "a generalised interval arithmetic", quite different to that presented in Section 3.2, for the purpose of reducing the dependency problem. However, the dependency problem can never be fully alleviated in polynomial time (unless $P = NP$), since obtaining exact bounds on all programmable functions would imply the global optimization problem could be solved on all programmable functions in polynomial time.

The quality of an interval inclusion function can be measured by looking at the width of the inclusions obtained (in terms of the width of the box it is taken over).

Definition 3.1.4 *The* width of an interval X *is* $w(X) = \max(X) - \min(X)$. *The* width of a box $\mathbf{X}$ *is* $w(\mathbf{X}) = \max_{i\in\{1,\ldots,n\}} w(X_i)$.

The following properties of widths are useful, see Alefeld and Herzberger [2] for the proofs.

Proposition 3.1.1 *Let X and Y be intervals. Then*

$$X \subseteq Y \Rightarrow w(X) \leq w(Y),$$

$$w(X \pm Y) = w(X) + w(Y)$$

and

$$w(X \cdot Y) \leq w(X)\max(|Y|) + \max(|X|)w(Y).$$

Natural inclusion functions, and inclusion functions in general, usually give inclusions of very large widths for boxes of large width. However, as the width of the box on which the natural inclusion functions is applied tends to zero, the widths of the inclusions given by the natural inclusion function tend to zero (implying the inclusion tends to the range of the function). Furthermore, the rate of convergence is $O(w(\mathbf{X})^\alpha)$, where $\alpha \in (0, 1]$, and $O(w(\mathbf{X}))$ if all the subexpressions of the function's expression are Lipschitz continuous. This slightly more general result than appears in the literature is summarised in the following theorem (cf. [2, 57]).

Theorem 3.1.1 *Let $f(\mathbf{x})$ be an expression of a programmable function, $f(\mathbf{X})$ be the expression for the natural inclusion function of $f(\mathbf{x})$ and suppose that $f(\mathbf{Y})$ is defined where $\mathbf{Y}$ is a box. Then, for all boxes $\mathbf{X} \subseteq \mathbf{Y}$, $w(f(\mathbf{X})) = O(w(\mathbf{X})^\alpha)$, for some $0 < \alpha \leq 1$. Furthermore, if all subexpressions of $f(\mathbf{x})$ are Lipschitz continuous, then α can be taken as one.*

Proof: For the trivial cases note that, if $f(\mathbf{x}) = x_i$, for some $i \in \{1, \ldots, n\}$, then $w(f(\mathbf{X})) = w(X_i) \leq w(\mathbf{X})$; and if $f(\mathbf{x}) = C$, for some $C \in \mathbb{R}$, then $w(f(\mathbf{x})) = 0 \leq w(\mathbf{X})$.

Now, suppose that $f(\mathbf{X}) = (g(\mathbf{X}) * h(\mathbf{X}))$ where $* \in \{+, -, \cdot, /\}$ and that the result holds for $g(\mathbf{X})$ and $h(\mathbf{X})$ (since $f(\mathbf{Y})$ is defined, both $g(\mathbf{Y})$ and $h(\mathbf{Y})$ are

defined). Since $f(\mathbf{Y})$ is defined, $0 \notin h(\mathbf{X})$ if $* = /$, for any $\mathbf{X} \subseteq \mathbf{Y}$, by the inclusion monotonicity property. Using Proposition 3.1.1 we have

$$\begin{aligned} w(g(\mathbf{X}) \pm h(\mathbf{X})) &= w(g(\mathbf{X})) + w(h(\mathbf{X})) = O(w(\mathbf{X})^\alpha) \text{ and} \\ w(g(\mathbf{X}) \cdot h(\mathbf{X})) &\leq w(g(\mathbf{X})) \max(|h(\mathbf{X})|) + \max(|g(\mathbf{X})|) w(h(\mathbf{X})) \\ &\leq w(g(\mathbf{X})) \max(|h(\mathbf{Y})|) + \max(|g(\mathbf{Y})|) w(h(\mathbf{X})) \\ &= O(w(\mathbf{X})^\alpha). \end{aligned}$$

For the case where $f(\mathbf{X}) = g(\mathbf{X})/h(\mathbf{X})$ note that

$$w(g(\mathbf{X})/h(\mathbf{X})) = w(g(\mathbf{X}) \cdot [1/\max(h(\mathbf{X})), 1/\min(h(\mathbf{X}))])$$

and

$$\begin{aligned} w([1/\max(h(\mathbf{X})), 1/\min(h(\mathbf{X}))]) &= 1/\min(h(\mathbf{X})) - 1/\max(h(\mathbf{X})) \\ &= \frac{\max(h(\mathbf{X})) - \min(h(\mathbf{X}))}{\min(h(\mathbf{X})) \max(h(\mathbf{X}))} \\ &\leq w(h(\mathbf{X}))/\min(|h(\mathbf{Y})|)^2 = O(w(h(\mathbf{X}))^\alpha). \end{aligned}$$

The result now follows from the multiplication rule above.

Finally, suppose that $f(\mathbf{X}) = r(h(\mathbf{X}))$, where r is an elementary function and the result holds for $h(\mathbf{X})$. Note that for all of the elementary functions r, if r is defined on the interval $[a, b]$ then $|r(x) - r(y)| \leq M|x - y|^\beta$, for some $M \geq 0$, $\beta \in (0, 1]$ and for all $x, y \in [a, b]$. Hence, for all $\mathbf{X} \in \mathbf{Y}$,

$$\begin{aligned} w(r(h(\mathbf{X}))) &= \max_{x \in h(\mathbf{X})} r(x) - \min_{x \in h(\mathbf{X})} r(x) \\ &\leq \max_{x,y \in h(\mathbf{X})} M|x - y|^\beta \\ &= w(h(\mathbf{X}))^\beta \\ &= O(w(\mathbf{X})^{\alpha\beta}). \end{aligned}$$

Furthermore, if all the subexpressions of $f(\mathbf{x})$ are Lipschitz continuous then $\alpha = \beta = 1$.

By induction the result holds for all programmable functions and if all the α's and β's are one, the final $\alpha = 1$. □

If the range of a function f over a box $\mathbf{X}$ is large, then the width of an interval inclusion of the range cannot be small. For this reason the following definitions are useful when comparing interval inclusion functions.

Definition 3.1.5 *The* excess width *of an inclusion f over a box $\mathbf{X}$ is the difference between the width of the inclusion and the width of the range of f over $\mathbf{X}$. The* relative width *of an inclusion f over a box $\mathbf{X}$ is the ratio of the width of the inclusion to the width of the range of f over $\mathbf{X}$.*

The convergence of the excess width is at least as good as the convergence of the width.

Corollary 3.1.1 *With the same definitions and conditions as Theorem 3.1.1, the* excess width *of $f(\mathbf{X})$ is $O(w(\mathbf{X}))^\alpha$, for some $\alpha \in (0,1]$ and α can be taken as one if all the subexpressions are Lipschitz.*

Proof: The excess width of $f(\mathbf{X})$ is bounded above by $w(f(\mathbf{X}))$ and below by zero. The result follows. □

3.1.1 Taylor Forms

Higher orders of convergence of the excess width, up to $O(w(\mathbf{X})^2)$, can be achieved by expressing the function in a Taylor expansion and using interval arithmetic to bound the derivatives. For instance, if $f \in C^1$ and $\mathbf{a} \in \mathbf{X}$ then for any point $\mathbf{x} \in \mathbf{X}$ there exists a point $\mathbf{c} \in \mathbf{X}$ such that

$$f(\mathbf{x}) = f(\mathbf{a}) + \nabla f(\mathbf{c})^T(\mathbf{x} - \mathbf{a}).$$

An interval inclusion of each of the entries in the gradient vector can be obtained by evaluating the natural inclusion functions of each of the partial derivatives. Writing this *interval gradient* so obtained as $\nabla f(\mathbf{X})$ it follows that

$$f(\mathbf{x}) \in f(\mathbf{a}) + \nabla f(\mathbf{X})^T(\mathbf{x} - \mathbf{a}),$$

for all $\mathbf{x} \in \mathbf{X}$. This interval inclusion function is known as the *first order Taylor form*. Similarly, if $f \in C^2$, the *second order Taylor form* can be written

$$f(\mathbf{x}) \in f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{X} - \mathbf{a}) + (\mathbf{X} - \mathbf{a})H(\mathbf{X})(\mathbf{X} - \mathbf{a})/2,$$

where $\mathbf{x} \in \mathbf{X}$ and $H(\mathbf{X})$ is an interval matrix obtained by evaluating the natural inclusion function of each of the second partial derivatives.

Both of these inclusion functions, and higher order Taylor forms, have (only) a quadratic order of convergence of the excess width [57]. Higher orders of convergence of the excess width using different expansions are also reported in [57]. In practice, only first, and sometimes second, order Taylor forms are used because of the high computational costs associated with higher orders of convergence.

Note that, in general the order of convergence of the width of the Taylor form inclusions is not necessarily any better than linear even if all subexpressions are Lipschitz continuous. Consider, for instance, $f(x) = x$ with exact inclusions of the first and second derivatives.

3.1.2 Conclusions

The way in which interval arithmetic has been conceived and defined leads naturally to inclusion functions defined on boxes. Consequently, global optimization algorithms which use interval arithmetic ([56, 25]) subdivide the domain of the function into boxes. In the next section interval arithmetic is generalised to non-box domains and it is shown how improved inclusions can be obtained by making use of the structure of the function's (possibly rewritten) expression, together with the structure of the domain.

3.2 Generalised Interval Arithmetic

There are a number of global optimization algorithms which find bounds on the objective function over simplexes rather than boxes (for instance, see [33]). In general, one might require an inclusion of function values over other domains. It seems sensible to extend the idea of interval inclusion functions (which are defined only on the class of boxes) to other (non-box) classes.

Definition 3.2.1 *Let $\mathcal{D}$ be a class of subsets of $\mathbb{R}^n$. A function $F : \mathcal{D} \to \mathbb{I}$ is an* inclusion function of $f : \cup\mathcal{D} \to \mathbb{R}$ over $\mathcal{D}$ *if for all $D \in \mathcal{D}$, $\mathbf{x} \in D \Rightarrow f(\mathbf{x}) \in F(D)$.*

To remain useful to global optimization two conditions should be met (see Chapter 5). Firstly, for any $\delta > 0$, the union of $D(\delta) = \{D \in \mathcal{D} | \text{diam}(D) \leq \delta\}$ must cover the domain of f. Secondly, the inclusion $F(D)$ must tend to the range of f over D as $\text{diam}(D)$ tends to zero.

By the above definition, the natural inclusion function of an expression of a programmable function $f(\mathbf{x})$ is an inclusion function of f over the class of all boxes in $\mathbb{R}^n$ (on which the natural inclusion function is defined). But also, an inclusion function F of a programmable function $f : \mathbb{R}^n \to \mathbb{R}$ over the set of all simplexes in $\mathbb{R}^n$ can be defined by $F(D) = f(\mathbf{X})$ where $\mathbf{X}$ is the smallest box containing the simplex D. Because $f(\mathbf{x}) \in f(\mathbf{X}) = F(D), \forall \mathbf{x} \in D$, F is an inclusion function for f over the simplexes. Such an example however loses all of the special structure of the simplex.

To make use of the structure of the function's expression $f(\mathbf{x})$ and the subdomains in $\mathcal{D}$, one can treat $f(\mathbf{x})$ as being built out of (possibly large) "base subexpressions".

Definition 3.2.2 *Let $f(\mathbf{x})$ be an expression of a programmable function and $\mathcal{B}$ be a set of subexpressions of $f(\mathbf{x})$ for which each occurrence of the independent variables of $f(\mathbf{x})$ is contained in exactly one subexpression in $\mathcal{B}$. Then $\mathcal{B}$ is called a set of* base subexpressions of $f(\mathbf{x})$.

For instance, as shown in Figure 3.1, the subexpressions, $\sin(x_1)(x_1 + x_2)$ and $\cos(x_2)(x_1 - x_2)$, form a set of base subexpressions of Expression (3.1).

If (preferably fast and exact) inclusion functions of the base subexpressions are known then, given $D \in \mathcal{D}$, inclusions for the base subexpressions on D can be found. With these inclusions, interval arithmetic can be used to find an inclusion for the entire function on D. The inclusion function so obtained is defined below.

Definition 3.2.3 *Let $\mathcal{D}$ be a class of subsets of $\mathbb{R}^n$, $f(\mathbf{x})$ be an expression of a programmable function and $\mathcal{B}$ be a set of base subexpressions of $f(\mathbf{x})$. The* natural generalised inclusion function $F : \mathcal{D} \to \mathbb{I}$ of $f(\mathbf{x})$ with base subexpressions $\mathcal{B}$ over $\mathcal{D}$, *is defined as follows: given $D \in \mathcal{D}$ replace the base subexpressions in $f(\mathbf{x})$ by their inclusions on D. Now, treating the constants, arithmetic operators and elementary functions as their interval counterparts, compute the resulting interval expression.*

For example, suppose we have just developed a new method to obtain fast exact inclusions over any box for functions of the form $r(x_i)(\sum_{j=1}^{n} c_j x_j)$, where r is a trigonometric function. Treating Expression (3.1) as an expression of a programmable function with base subexpressions $\sin(x_1)(x_1 + x_2)$ and $\cos(x_2)(x_1 - x_2)$,

and using our new method, working to one decimal place, the natural generalised inclusion function of $f(\mathbf{x})$ with these base subexpression on the box $[0,2] \times [-1,4]$ gives

$$F([0,2],[-1,4]) = [-0.3,5.7] + [-0.6,3.3] = [-0.9,9.0].$$

Note that this interval is a valid inclusion of f on this box and is tighter than that given by the natural inclusion function above. Furthermore, if we develop a method to find inclusions for these base subexpressions over $\mathcal{D}$, where $\mathcal{D}$ is any class of subdomains in R^n, then a natural generalised inclusion function of $f(\mathbf{x})$ with these subexpressions over $\mathcal{D}$ could be obtained similarly.

The following theorem shows that natural generalised inclusion functions are indeed inclusion functions.

Theorem 3.2.1 *Let $\mathcal{D}$ be a class of subsets of R^n, f be a programmable function with expression $f(\mathbf{x})$ and $\mathcal{B}$ be a set of base subexpressions of $f(\mathbf{x})$. The natural generalised inclusion function of $f(\mathbf{x})$ with base subexpressions $\mathcal{B}$ over $\mathcal{D}$ is an inclusion function of f over $\mathcal{D}$.*

Proof: Let $f(\mathbf{x})$ be an expression of a programmable function with base subexpressions $\mathcal{B}$. If $f(\mathbf{x}) = C$, where $C \in \mathbb{R}$, or $f(\mathbf{x}) \in \mathcal{B}$, then the result follows immediately.

Now, suppose that $f(\mathbf{x}) = (g(\mathbf{x}) * h(\mathbf{x}))$, where $* \in \{+. - \cdot, /\}$, and the result holds for $g(\mathbf{x})$ and $h(\mathbf{x})$. Let G and H be the natural generalised inclusion functions of $g(\mathbf{x})$ and $h(\mathbf{x})$ with base subexpressions $\mathcal{B}$ over $\mathcal{D}$, respectively. Then the natural generalised inclusion function of $f(\mathbf{x})$ with base subexpressions $\mathcal{B}$ over $\mathcal{D}$ is given by $(G(D) * H(D))$, where $D \in \mathcal{D}$. If $\mathbf{x} \in D$, then

$$f(\mathbf{x}) = (g(\mathbf{x}) * h(\mathbf{x})) \in \{(g(\mathbf{x}) * h(\mathbf{y}))|\mathbf{x},\mathbf{y} \in D\} \subseteq (G(D) * H(D)).$$

The case where $f(\mathbf{x}) = r(h(\mathbf{x}))$ is similar.

Since the leaves of the expression tree of $f(\mathbf{x})$ are either a constant or contained in a base subexpression, the results follows by induction. □

As well as allowing inclusion functions over more general classes of domains, if exact inclusion functions are used to find inclusions of the base subexpressions of $f(\mathbf{x})$ over the boxes, then the natural generalised inclusion function can provide tighter inclusions than the natural inclusion function.

Proposition 3.2.1 *Let $\mathcal{D}$ be the set of all boxes and $f(\mathbf{x})$ be an expression of a programmable function with base subexpressions $\mathcal{B}$ for which exact inclusion functions over $\mathcal{D}$ are known. If $F(\mathbf{X})$ is the natural generalised inclusion function of $f(\mathbf{x})$ with base subexpressions $\mathcal{B}$ over $\mathcal{D}$ and $f(\mathbf{X})$ is the natural inclusion function of $f(\mathbf{x})$ then $F(\mathbf{X}) \subseteq f(\mathbf{X})$, where $\mathbf{X}$ is a box, as long as $f(\mathbf{X})$ is defined.*

Proof: Note that, if $f(\mathbf{x}) \in \mathcal{B}$ and $f(\mathbf{X})$, where $\mathbf{X}$ is a box, is defined then $f(\mathbf{X}) \subseteq F(\mathbf{X})$ since $F(\mathbf{X})$ is as tight as possible. The result now follows from the inclusion monotonicity property of interval arithmetic. □

In order to use generalised interval arithmetic effectively, one should simultaneously consider possible expressions of f, possible sets of base subexpressions, the efficiency of the inclusion functions for the base subexpressions and possible classes of subsets of R^n. In the next section, the class of subsets is restricted to the simplexes or the boxes and only exact inclusion functions for the subexpressions are considered.

3.3 Examples

The following examples demonstrate how generalised interval arithmetic is used. The first example shows how using larger base subexpressions can lead to tighter inclusions on simplexes. In the second example the natural inclusion function is replaced with a natural generalised inclusion function, leading to dramatic improvements in the performance of a global optimization algorithm. The final example gives natural generalised inclusion functions for a function, and its first and second partial derivatives, which arises in practice.

Example 3.3.1 The first example shows how generalised interval arithmetic can be used on non-box domains (simplexes) and how using larger base subexpressions can lead to better inclusions, at the cost of more preparation work and computation time. Consider finding inclusion functions for a function of the form

$$f(\mathbf{x}) = \prod_i \sum_j a_{ij} x_j$$

over the set of all simplexes. Specifically, we will look at the function given by

$$f(\mathbf{x}) = (x - y + z)(2x + 2y + z)(x + 2y + 2z)(2x + y + 2z)(x - y - z)(x + y - z),$$

where $\mathbf{x} = (x, y, z)^T$, and find inclusions over the simplex S with vertices at the points $(1, -2, 1)$, $(1, -2, 3)$, $(3, 1, -1)$, and $(1, 1, 1)$.

Three different sets of base subexpressions are used to obtain three different natural generalised inclusion functions for f over the simplexes. First, the 18 occurances of the independent variables are taken to be the set of base subexpressions leading to the simple minded inclusion function over simplexes discussed above. Next, the six linear factors are taken to be the set of base subexpression. Finally, the first four linear factors are treated as one base subexpression, and together with the remaining two linear subexpressions, form the last set of base subexpressions. Figure 3.2 shows the expression tree for f and three sets of base subexpressions used.

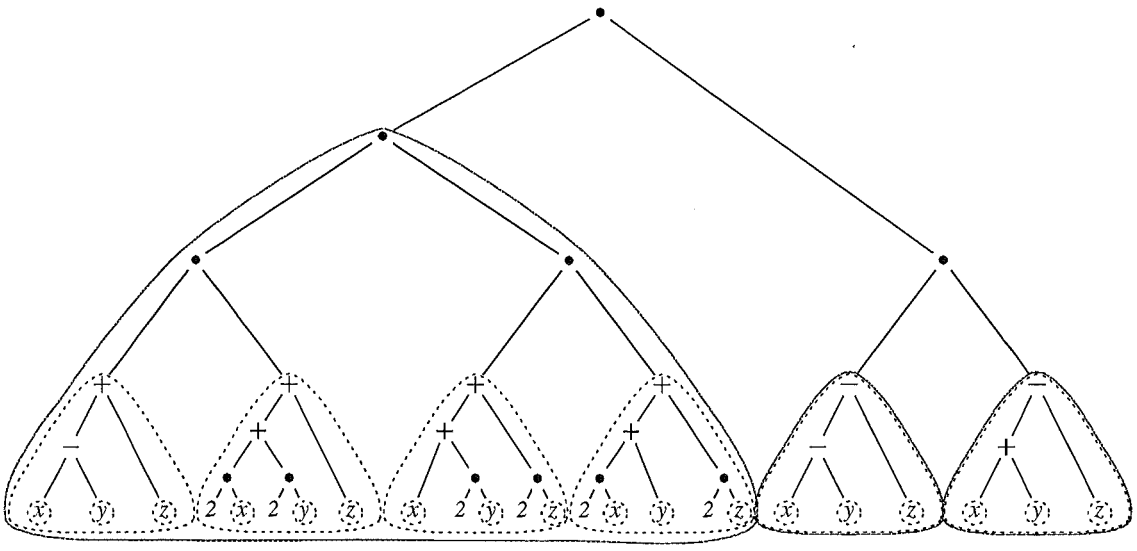

Figure 3.2: The expression tree for $f(x)$ with three nested sets of base subexpressions.

Methods to find inclusion function for these sets of subexpressions over the simplexes contained in S are described below. The inclusions given by natural generalised inclusion functions of $f(\mathbf{x})$ with each of the three sets of base subexpressions on S were computed with the help of MATLAB.

- An exact inclusion function for the projection functions over the simplexes can be found by projecting the simplex onto the axes. In particular, projecting S onto the axes, we find that $x \in [1, 3]$, $y \in [-2, 1]$ and $z \in [-3, 1]$. Now, using the natural generalised inclusion function of $f(\mathbf{x})$ with these base

subexpressions, we can compute

$$\begin{aligned} F(S) &= ([1,3]-[-2,1]-[-3,1])\cdot([1,3]-[-2,1]+[-3,1])\cdot \\ &\quad ([1,3]+[-2,1]-[-3,1])\cdot(2[1,3]+2[-2,1]+[-3,1])\cdot \\ &\quad ([1,3]+2[-2,1]+2[-3,1])\cdot(2[1,3]+[-2,1]+2[-3,1]) \\ &= [-302016, 377520]. \end{aligned}$$

This inclusion requires approximately the same number of floating point operations as for two functions evaluations.

- An exact inclusion function for the linear functions over the simplexes can be found by taking the minimum and maximum function evaluations over the $n+1$ vertices of the simplex. In particular, by computing exact inclusions for each of the linear subexpressions of $f(\mathbf{x})$, the natural generalised inclusion function for $f(\mathbf{x})$ with these base subexpressions gives the much improved inclusion of f on S of

$$F(S) = [-1,3]\cdot[1,6]\cdot[-4,5]\cdot[1,7]\cdot[1,5]\cdot[3,6] = [-15120, 18900].$$

The same number of floating point operations as for four functions evaluations were required.

- An exact inclusion function for the subexpression made up of the first four linear factors of $f(\mathbf{x})$ over any simplex contained in S can be found by taking the minimum function evaluation at the vertices of the simplex to find the lower bound and applying a local maximization algorithm to find the upper bound. To see this, consider the change of coordinates

$$\begin{aligned} \hat{x} &= x - y + z \\ \hat{y} &= 2x + 2y + z \\ \hat{z} &= x + 2y + 2z \\ \hat{w} &= 2x + y + 2z. \end{aligned}$$

Note that the transformed simplex S lies completely in the positive quadrant in the new coordinates (vertices at $(3,1,1,3)$, $(6,1,3,6)$, $(1,7,3,5)$ and $(1,5,5,5)$) and, by considering the second partial derivatives, $\log(\hat{x}\hat{y}\hat{z}\hat{w})$ can be shown to

be convex in the positive quadrant. It follows that, for any simplex contained in S, the subexpression made up of the first four linear factors of $f(\mathbf{X})$ has a unique local maximum and the minimum is at a vertex of the simplex. (For a more general discussion of the related topic of "geometric programming" the reader is referred to [13].)

In particular, using this method to find an inclusion of the first four linear factors and using the same inclusion as above for the remaining two linear factors, the natural generalised inclusion function of $f(\mathbf{x})$ with these base subexpressions gives an inclusion of f on S of

$$F(S) = [9, 331.7] \cdot [-1, 3] \cdot [-4, 5] = [-3980, 4975].$$

This required approximately the same number of floating point operations as for 750 function evaluations (MATLAB's optimization toolbox function `constr` was used for the local optimization).

The above results, together with the estimated relative widths of the inclusions (based on the estimated range of f found by a multi start algorithm with 100 starting-points) and the estimated preparation times, are summarised in the following table.

Base subexpressions	Inclusion	Rel. Width	Evaluations	Preparation
First set	[-302016,377520]	354	2	2 minutes
Second set	[-15120,18900]	18	4	5 minutes
Third set	[-3980,4975]	5	750	15 minutes

Table 3.1: Summary of results.

As can be seen, improved inclusions can be obtained by using larger and more complicated base subexpression of $f(\mathbf{x})$. The cost, however, is more computation and preparation time. The estimated preparation time presented in the table is based on the time it took the author to recognise and make use of subexpressions in the expression of f; it does not take into account the years of training required to allow the author to identify useful subexpressions.

Example 3.3.2 This example illustrates how generalised interval arithmetic can improve the overall performance of global optimization algorithms. In this example, the additional computational cost of finding tighter inclusions pays off by reducing the number of function evaluations required by the global optimization algorithm. Consider the "Peaks" function given by the expression

$$\begin{aligned} f(x,y) &= 3(1-x)^2\exp(-x^2-(y+1)^2) - 10(x/5-x^3-y^5)\exp(-x^2-y^2) \\ &\quad -\frac{1}{3}\exp(-(x+1)^2-y^2) \end{aligned}$$

and defined on $[-3,3]^2$, where $\mathbf{x}=(x,y)$.

By studying the structure of the function's expression, we can find a suitable set of base subexpressions of a slightly rewritten expression of f. Note that, if the first term is bounded by interval arithmetic, then the dependency between x in the quadratic factor and x in the exponential is not taken into account. This is likely to lead to poor inclusions for large boxes containing the origin since the inclusion of the quadratic term will be multiplied by one.

However, the first term of $f(\mathbf{x})$ can be rewritten as

$$3(1-x)^2\exp(-x^2)\exp(-(y+1)^2).$$

The first part of this subexpression, $3(1-x)^2\exp(-x^2)$, has turning points at 1 and $1/2\pm\sqrt{5}/2$. Therefore an exact inclusion of the first part over an interval can be found by taking the minimum and maximum of the function value at the endpoints and the stationary points in the interval. Since y is independent of $3(1-x)^2\exp(-x^2)$ an inclusion of the whole first term can be computed exactly by using interval arithmetic to multiply the inclusions of the first part and $\exp(-(y+1)^2)$. (Interval arithmetic gives an exact inclusion of $\exp(-(y+1)^2)$ since y appears only once.)

The second term of $f(\mathbf{x})$,

$$-10(x/5-x^3-y^5)\exp(-x^2-y^2),$$

is similar except that the polynomial factor is not univariate. However, it can be rewritten as

$$-10(x/5-x^3)\exp(-x^2)\exp(-y^2)+10y^5\exp(-y^2)\exp(-x^2)$$

and for each of these terms, exact inclusions can be found similarly to the above.

For the final term of $f(\mathbf{x})$,

$$-\frac{1}{3}\exp(-(x+1)^2 - y^2),$$

interval arithmetic already gives exact inclusions since each variable occurs only once.

Thus, a set of base subexpressions of a slightly rewritten expression of f can be formed from $3(1-x)^2\exp(-x^2)\exp(-(y+1)^2)$, $-10(x/5-x^3)\exp(-x^2)\exp(-y^2)$ and $10y^5\exp(-y^2)\exp(-x^2)$, together with the remaining occurrences of the independent variables. The natural generalised inclusion function of the rewritten expression of f with these subexpressions takes about 20% longer to evaluate than the natural inclusion function of the original expression.

However, the additional effort for computing this natural generalised inclusion function is greatly rewarded when used in a global optimization algorithm. To illustrate this, the Ichida-Fujii Algorithm [34] was implemented in C++ and run on the Peaks function using the natural inclusion function and the natural generalised inclusion function described above. Table 3.2 summarises the results.

ϵ	Natural	Generalised
1	182 (0.11s)	9 (0.01s)
10^{-1}	1450 (0.92s)	12 (0.01s)
10^{-2}	12762 (8.54s)	18 (0.01s)
10^{-3}	135311 (97.7s)	35 (0.03s)
10^{-4}	1357761 (1055s)	131 (0.11s)
10^{-5}	—	880 (0.72s)
10^{-6}	—	8509 (7.17s)
10^{-7}	—	84766 (76.6s)
10^{-8}	—	791111 (753s)

Table 3.2: Iterations (CPU times) for the Ichida-Fujii Algorithm on the Peaks function using the natural inclusion function and a natural generalised inclusion function.

As can be seen, the number of iterations required by the algorithm when using the natural generalised inclusion function is dramatically reduced in comparison to using the natural inclusion function. This results from tighter inclusions, meaning subdomains are removed at earlier stages and do not have to be subdivided.

Example 3.3.3 The final example shows how generalised interval arithmetic might be applied to functions arising from "real" problems. An inclusion function using generalised interval arithmetic is found for a function and its first and second partial derivatives arising from an application in mathematical biology discussed further in Appendix A. Let f be given by

$$f(\mathbf{x}) = f_1 f_2 f_3 f_4 f_5 f_6,$$

where

$$\begin{aligned}
f_1 &= (1 + x_1x_2 + x_3x_4 + x_1x_3x_5 + x_2x_3x_5 + x_1x_4x_5 + x_2x_4x_5 + x_1x_2x_3x_4),\\
f_2 &= (1 - x_1x_2 - x_3x_4 - x_1x_3x_5 + x_2x_3x_5 + x_1x_4x_5 - x_2x_4x_5 + x_1x_2x_3x_4),\\
f_3 &= (1 - x_1x_2 + x_3x_4 - x_1x_3x_5 + x_2x_3x_5 - x_1x_4x_5 + x_2x_4x_5 - x_1x_2x_3x_4),\\
f_4 &= (1 - x_1x_2 + x_3x_4 + x_1x_3x_5 - x_2x_3x_5 + x_1x_4x_5 - x_2x_4x_5 - x_1x_2x_3x_4),\\
f_5 &= (1 + x_1x_2 - x_3x_4 - x_1x_3x_5 - x_2x_3x_5 + x_1x_4x_5 + x_2x_4x_5 - x_1x_2x_3x_4) \text{ and}\\
f_6 &= (1 + x_1x_2 - x_3x_4 + x_1x_3x_5 + x_2x_3x_5 - x_1x_4x_5 - x_2x_4x_5 - x_1x_2x_3x_4),
\end{aligned}$$

and defined on $[0,1]^5$. The partial derivatives of $f(\mathbf{x})$ are given by

$$\frac{\partial}{\partial x_i} f(\mathbf{x}) = \sum_{k=1}^{6} \frac{\partial f_k}{\partial x_i} \prod_{l \neq k} f_l.$$

The second partial derivatives of $f(\mathbf{x})$ are given by

$$\frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x}) = \sum_{k=1}^{6} \sum_{l \neq k} \frac{\partial f_k}{\partial x_i} \frac{\partial f_l}{\partial x_j} \prod_{m \neq k,l} f_m + \sum_{k=1}^{6} \frac{\partial f_k^2}{\partial x_i \partial x_j} \prod_{l \neq k} f_l.$$

Observe that the subexpressions f_k, $k = 1, \ldots, 6$, are *multilinear*, that is, linear when all but one of the variables are fixed. Also, the partial and mixed second partial derivatives have many multilinear subexpressions.

Exact inclusion functions for multilinear functions over the boxes can be found by observing that the maximum and minimum can only occur at the vertices. Thus tighter inclusions can be found by using the natural generalised inclusion functions with multilinear base subexpressions rather than the natural inclusion function but requires evaluating the multilinear subexpressions at each of 32 vertices.

3.4 Conclusions

Generalised interval arithmetic provides a method to tailor inclusion functions to provide tighter inclusions of real valued functions. It can be used whenever interval arithmetic is used, including finding improved inclusions of gradients and Hessians for use in Taylor expansions. It also allows other (non-box) classes of domain to be considered, opening the way to new global optimization algorithms.

As demonstrated by the second example, when generalised interval arithmetic is used in global optimization great improvements can occur. This could mean the difference between solving a problem (in reasonable time and memory) and not solving it.

To find good inclusion functions using generalised interval arithmetic, however, the class of subdomains, possible expressions for the function and possible sets of base subexpressions must be considered simultaneously. At present this must be performed by hand and may require considerable effort. For this reason, it is recommended for one-off problems to use generalised interval arithmetic only if attempts to solve the problem without it fail. When considering many similar global optimization problems, a few representative examples could be chosen and the benefits (if any) of using generalised interval arithmetic determined through a process of trial and error.

In the future, a systematic approach to the use of generalised interval arithmetic may be possible, allowing for the automation of this process.

Chapter 4

Interval and bounding Hessians

This chapter describes how to find certain upper and lower bounds on the second derivatives of a real-valued function over a subdomain. These "bounding Hessians" are required, (although no method for obtaining them is provided), by Baritompa and Cutler's [6] extensions to the Breiman and Cutler algorithm [8] for global optimization.

Bounding Hessians are also useful for a number of other global optimization methods. They can be used to rewrite functions as d.c. functions, to reformulate general problems as GOP [17], to find convex under-estimators of functions (for an extension to the αBB algorithm of Androulakis *et al.*[4] for instance) and to find bounds on the eigenvalues of the Hessian.

In the previous chapter it was shown that if an expression for a function is known then (generalised) interval arithmetic can provide inclusions of the function over a subdomain. If expressions for the second partial derivatives are known, then (generalised) interval arithmetic can provide "interval Hessians" of the function over a subdomain. This chapter concentrates on the further step of obtaining "bounding Hessians" from "interval Hessians".

4.1 Definitions

Suppose that $f : D \to \mathbb{R}$ is a C^2 function, where D is a compact convex subset of $\mathbb{R}^n$ with non-empty interior. The Hessian of f at $\mathbf{a} \in D$ is the symmetric matrix

whose entries are the second partials of f at $\mathbf{a}$,

$$H_f(\mathbf{a}) = \left[\begin{array}{cccc} \partial f^2/\partial^2 \mathbf{x}_1 & \partial f^2/\partial \mathbf{x}_1 \partial \mathbf{x}_2 & \cdots & \partial f^2/\partial \mathbf{x}_1 \partial \mathbf{x}_n \\ \partial f^2/\partial \mathbf{x}_2 \partial \mathbf{x}_1 & \partial f^2/\partial^2 \mathbf{x}_2 & \cdots & \partial f^2/\partial \mathbf{x}_2 \partial \mathbf{x}_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial f^2/\partial \mathbf{x}_n \partial \mathbf{x}_1 & \partial f^2/\partial \mathbf{x}_n \partial \mathbf{x}_2 & \cdots & \partial f^2/\partial^2 \mathbf{x}_n \end{array} \right]_{\mathbf{x} = \mathbf{a}}.$$

An interval matrix $\mathcal{H} \in \mathbb{I}^{n \times n}$ is an *interval Hessian of* f if $H_f(\mathbf{a}) \in \mathcal{H}$, for all $\mathbf{a} \in D$. The interval matrix $\mathcal{H}$ is an *interval Hessian of a class of* C^2 *functions* if it is an interval Hessian of each function in the class. The largest class of C^2 functions for which $\mathcal{H}$ is an interval Hessian is denoted $C(\mathcal{H})$.

Interval Hessians can be obtained by finding inclusions of the second partial derivatives of f over D. Note that, if an interval Hessian $\mathcal{H}$ of f is not symmetric then replacing the ijth and jith entries with their intersection gives a symmetric interval Hessian of f over D (this is valid since the Hessian is symmetric). Hansen suggests the use of an upper triangular form for the interval Hessians "for reasons related to the use of interval analysis" [24]. Functionally, the upper triangular form is equivalent to the symmetric form. In this thesis symmetric interval Hessians are used for simplicity.

A symmetric matrix U is an *upper Hessian of* f if, for all $\mathbf{a}, \mathbf{x} \in D$,

$$f(\mathbf{x}) \leq f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x} - \mathbf{a}) + (\mathbf{x} - \mathbf{a})^T U(\mathbf{x} - \mathbf{a})/2. \tag{4.1}$$

A symmetric matrix U is an *upper Hessian of* $C(\mathcal{H})$ if U is an upper Hessian of every $f \in C(\mathcal{H})$. A symmetric matrix L is a *lower Hessian of* f if $-L$ is an upper Hessian of $-f$ and is a *lower Hessian of* $C(\mathcal{H})$ if L is a lower Hessian of every $f \in C(\mathcal{H})$. Collectively upper and lower Hessians are called *bounding Hessians.*

4.2 Obtaining Bounding Hessians from Interval Hessians

The results are given only for upper Hessians. Similar results can be obtained for lower Hessians. The first theorem gives a useful equivalent condition for upper Hessians.

Theorem 4.2.1 *A symmetric matrix U is an upper Hessian of f if and only if $U - H_f(\mathbf{a})$ is positive semi-definite for all $\mathbf{a} \in D$.*

Proof: Fix $\mathbf{a}$ and $\mathbf{x}$ in D. Since f is C^2 and D is convex we can write $f(\mathbf{x})$ as a first order Taylor expansion plus error term. That is,

$$f(\mathbf{x}) = f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x} - \mathbf{a}) + (\mathbf{x} - \mathbf{a})^T H_f(\mathbf{c})(\mathbf{x} - \mathbf{a})/2$$

for some $\mathbf{c} \in D$. If $U - H_f(\mathbf{c})$ is positive semi-definite then

$$\begin{aligned} f(\mathbf{x}) &\leq f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x} - \mathbf{a}) + (\mathbf{x} - \mathbf{a})^T H_f(\mathbf{c})(\mathbf{x} - \mathbf{a})/2 \\ &\quad +(\mathbf{x} - \mathbf{a})^T(U - H_f(\mathbf{c}))(\mathbf{x} - \mathbf{a})/2 \\ &= f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x} - \mathbf{a}) + (\mathbf{x} - \mathbf{a})^T U(\mathbf{x} - \mathbf{a})/2. \end{aligned}$$

For the converse, suppose that U is an upper Hessian and fix $\mathbf{a}$ in the interior of D. Note that $U - H_f(\mathbf{a})$ is the Hessian of

$$(f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x} - \mathbf{a}) + (\mathbf{x} - \mathbf{a})^T U(\mathbf{x} - \mathbf{a})/2) - f(\mathbf{x})$$

at $\mathbf{a}$. This function is less than or equal to zero for all $\mathbf{x}$ and equal to zero at $\mathbf{a}$, so $\mathbf{a}$ is a local minimizer. It follows that $U - H_f(\mathbf{a})$ is positive semi-definite. Positive semi-definiteness of $U - H_f(\mathbf{a})$ on the boundary follows from continuity of H_f. □

Finding upper Hessians using the above result is difficult in general. However, if an interval Hessian of f is known, then the following result provides a simple test of when a symmetric matrix is an upper Hessian of the class $C(\mathcal{H})$. It establishes that the constant and linear terms of (4.1) need not be considered.

Lemma 4.2.1 *Let $\mathcal{H}$ be a symmetric interval matrix. A symmetric matrix U is an upper Hessian of $C(\mathcal{H})$ if and only if $\mathbf{y}^T U\mathbf{y} \geq \max(\mathbf{y}^T\mathcal{H}\mathbf{y}), \forall \mathbf{y} \in \mathbb{R}^n$.*

Proof: It follows from the first order Taylor expansion plus error term that, for all $\mathbf{a}, \mathbf{x} \in D$ and $f \in C(\mathcal{H})$,

$$f(\mathbf{x}) \in f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x} - \mathbf{a}) + (\mathbf{x} - \mathbf{a})^T\mathcal{H}(\mathbf{x} - \mathbf{a})/2. \tag{4.2}$$

Thus, if $\mathbf{y}^T U\mathbf{y} \geq \max(\mathbf{y}^T\mathcal{H}\mathbf{y}), \forall \mathbf{y} \in \mathbb{R}^n$, it follows that, for all $\mathbf{a}, \mathbf{x} \in D$,

$$f(\mathbf{x}) \leq f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x} - \mathbf{a}) + (\mathbf{x} - \mathbf{a})^T U(\mathbf{x} - \mathbf{a})/2.$$

Conversely, suppose that there exists $\mathbf{y} \in \mathbb{R}^n$ such that $\mathbf{y}^T U \mathbf{y} < \max(\mathbf{y}^T \mathcal{H} \mathbf{y})$. Then, as D has non-empty interior, we can choose $\mathbf{a}, \mathbf{x} \in D$ such that $\mathbf{x} - \mathbf{a}$ is a positive multiple of $\mathbf{y}$ and it follows that $(\mathbf{x}-\mathbf{a})^T U (\mathbf{x}-\mathbf{a}) < \max((\mathbf{x}-\mathbf{a})^T \mathcal{H} (\mathbf{x}-\mathbf{a}))$. Now, $\max((\mathbf{x}-\mathbf{a})^T \mathcal{H}(\mathbf{x}-\mathbf{a})) = (\mathbf{x}-\mathbf{a})^T M (\mathbf{x}-\mathbf{a})$, where $M = [m_{ij}]$,

$$m_{ij} = \begin{cases} b_{ij} & \text{if } y_i y_j \geq 0 \\ a_{ij} & \text{if } y_i y_j < 0 \end{cases}$$

and $[a_{ij}, b_{ij}]$ is the $ij-$th entry in $\mathcal{H}$. Let $f(\mathbf{x}) = (\mathbf{x}-\mathbf{a})^T M (\mathbf{x}-\mathbf{a})/2$. Then $f(\mathbf{x}) \in C(\mathcal{H})$, $f(\mathbf{a}) = 0$, and $\nabla f(\mathbf{a}) = 0$. Therefore,

$$\begin{aligned} f(\mathbf{a}) + \nabla f(\mathbf{a})^T (\mathbf{x}-\mathbf{a}) + (\mathbf{x}-\mathbf{a})^T U (\mathbf{x}-\mathbf{a})/2 &= (\mathbf{x}-\mathbf{a})^T U (\mathbf{x}-\mathbf{a})/2 \\ &< (\mathbf{x}-\mathbf{a})^T M (\mathbf{x}-\mathbf{a})/2 \\ &= f(\mathbf{x}) \end{aligned}$$

so U is not an upper Hessian. □

Using the interval Hessian in Taylor's theorem, as in Equation (4.2), provides upper and lower envelopes of f. These envelopes are piecewise quadratic functions, whose Hessians depend on the quadrant in which $\mathbf{x} - \mathbf{a}$ lies (see the definition of $M(y)$ in the proof of Theorem 4.2.2). Unfortunately, despite the fact that a single evaluation of the envelope can be found in $O(n^2)$ time, these envelopes are difficult to use directly, in a covering method for instance, due to the intractable number of quadrants (2^n).

However, if we had "good" bounding Hessians that define *quadratic (upper and lower) envelopes of* f, given by the right hand side of equation (4.1), then these could be used directly, for instance, in Baritompa and Cutler's algorithm. The following theorem provides an $O(n^2)$ method of obtaining bounding Hessians associated with an interval Hessian.

Theorem 4.2.2 *Given a symmetric interval Hessian* $\mathcal{H}$ *with entries* $[a_{ij}, b_{ij}]$, $U = [u_{ij}]$ *is an upper Hessian of* $C(\mathcal{H})$, *where*

$$u_{ij} = \begin{cases} b_{ii} + 1/4 \sum_{k \neq i}^{n} (b_{ki} - a_{ki}) + (b_{ik} - a_{ik}) & i = j \\ (a_{ij} + b_{ij})/2 & i \neq j. \end{cases} \tag{4.3}$$

Furthermore, if all the components of $\mathbf{y} \in \mathbb{R}^n$ *are equal in magnitude, then* $\mathbf{y}^T U \mathbf{y} = \max(\mathbf{y}^T \mathcal{H} \mathbf{y})$.

Proof: Fix $\mathbf{y} \in \mathbb{R}^n$. Then $\max(\mathbf{y}^T\mathcal{H}\mathbf{y}) = \mathbf{y}^T M(\mathbf{y})\mathbf{y}$, where $M(\mathbf{y}) = [m_{ij}(\mathbf{y})]$,

$$m_{ij}(\mathbf{y}) = \begin{cases} b_{ij} & \text{if } y_i y_j \geq 0 \\ a_{ij} & \text{if } y_i y_j < 0 \end{cases}$$

The diagonal elements of $M(\mathbf{y})$ are b_{ii} because $y_i^2 \geq 0$. Consider an off diagonal element $m_{ij}(\mathbf{y}), i \neq j$. Note that

$$(a_{ij} + b_{ij})y_i y_j/2 + (b_{ij} - a_{ij})(y_i^2 + y_j^2)/4 \tag{4.4}$$

$$\geq \begin{cases} ((a_{ij} + b_{ij})/2 + (b_{ij} - a_{ij})/2)y_i y_j, & \text{if } y_i y_j \geq 0 \\ ((a_{ij} + b_{ij})/2 - (b_{ij} - a_{ij})/2)y_i y_j, & \text{if } y_i y_j < 0 \end{cases} \tag{4.5}$$

$$= m_{ij}(\mathbf{y})y_i y_j. \tag{4.6}$$

Summing (4.4) and (4.6) over all $i \neq j$ gives

$$\mathbf{y}^T U\mathbf{y} - \sum_{i=1}^{n} b_{ii}y_i^2 \geq \mathbf{y}^T M(\mathbf{y})\mathbf{y} - \sum_{i=1}^{n} b_{ii}y_i^2,$$

which implies $\mathbf{y}^T U\mathbf{y} \geq \mathbf{y}^T M(\mathbf{y})\mathbf{y}$. So, for each $\mathbf{y}$, $\mathbf{y}^T U\mathbf{y} \geq \max(\mathbf{y}^T\mathcal{H}\mathbf{y})$ and, by Lemma 4.2.1, U is an upper Hessian.

Furthermore, equality in (4.5) holds if $y_i = \pm y_j, \forall i, j \in \{1, \ldots, n\}$. Thus, $\mathbf{y}^T U\mathbf{y} = \mathbf{y}^T M(\mathbf{y})\mathbf{y} = \max(\mathbf{y}^T\mathcal{H}\mathbf{y})$. □

The upper Hessian associated with $\mathcal{H}$ obtained by this Theorem is denoted $U(\mathcal{H})$. A similar result can be obtained for lower Hessians associated with $\mathcal{H}$. These are denoted $L(\mathcal{H})$.

4.3 Optimality

This section gives the optimal properties of $U(\mathcal{H})$.

If f and ∇f are evaluated at $\mathbf{a}$, Equation (4.2) gives a piecewise quadratic upper envelope which provides the lowest upper bound of every function in $C(\mathcal{H})$ at each point in the domain. The last part of Theorem 4.2.2, shows that the corresponding quadratic upper envelope of $U(\mathcal{H})$ touches this envelope along a line in each quadrant about $\mathbf{a}$, (see Figure 4.1). Using this, two corollaries showing optimality follow.

The first corollary shows that $U(\mathcal{H})$ minimizes the trace over all possible upper Hessians of $C(\mathcal{H})$. Recall that trace is also the sum of the eigenvalues. First, the following lemma is established.

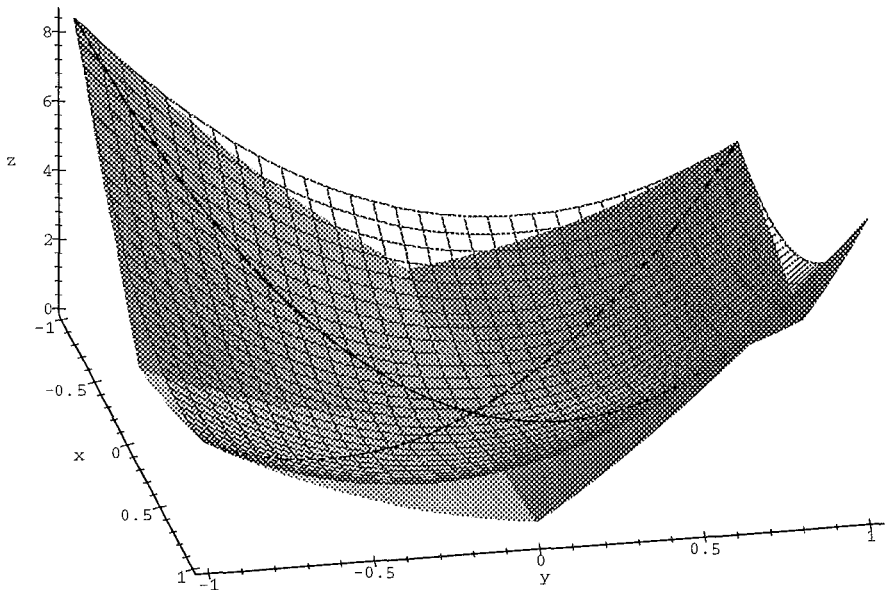

Figure 4.1: A quadratic envelope (wire-frame surface) obtained through Theorem 4.2.2 sitting on top of the piecewise quadratic envelope given by Equation (4.2) (solid surface). The envelopes touch along the lines $x = \pm y$.

Lemma 4.3.1 *Given any matrix* $A \in \mathbb{R}^{n\times n}$, *there exists* $\mathbf{y} \in \{-1,1\}^n$ *such that* $\mathbf{y}^T A\mathbf{y} \leq tr(A)$.

Proof: The proof is by induction on n. For $n = 1$, letting $y = 1$ gives the result. Now, suppose that the lemma is true for $n = k-1$ and that $A \in \mathbb{R}^{k\times k}$. Then, for any $\mathbf{y} \in \mathbb{R}^n$,

$$\begin{aligned}\mathbf{y}^T A\mathbf{y} &= \sum_{i=1}^{k}\sum_{j=i}^{k} y_i y_j a_{ij} \\ &= \sum_{i=1}^{k-1}\sum_{j=1}^{k-1} y_i y_j a_{ij} + y_k \sum_{i=1}^{k-1} y_i(a_{ki}+a_{ik}) + a_{kk}\end{aligned}$$

Thus, $y_1, \ldots, y_{k-1} \in \{-1,1\}$ can be chosen, by the induction hypothesis, so that

$$\sum_{i=1}^{k-1}\sum_{j=1}^{k-1} y_i y_j a_{ij} \leq \sum_{i=1}^{k-1} a_{ii},$$

and $y_k \in \{-1,1\}$ can be chosen so that

$$y_k \sum_{i=1}^{k-1} y_i(a_{ki}+a_{ik}) + a_{kk} \leq a_{kk}.$$

Thus, the induction hypothesis is true for $n = k$. □

The first corollary can now be proven.

Corollary 4.3.1 *Given a symmetric interval Hessian* $\mathcal{H}$, *the associated upper Hessian obtained from Theorem 4.2.2,* $U(\mathcal{H})$ *minimizes the trace over all upper Hessians of* $C(\mathcal{H})$.

Proof: Let U' be a symmetric matrix and suppose $\mathrm{tr}(U') < \mathrm{tr}(U(\mathcal{H}))$. Then $\mathrm{tr}(U' - U(\mathcal{H})) = \mathrm{tr}(U') - \mathrm{tr}(U(\mathcal{H})) < 0$, so , by Lemma 4.3.1, there exists a vector $\mathbf{y} \in \{-1,1\}^n$ such that

$$\mathbf{y}^T(U' - U(\mathcal{H}))\mathbf{y} \leq \mathrm{tr}(U' - U(\mathcal{H})) < 0. \tag{4.7}$$

Thus $\mathbf{y}^T U'\mathbf{y} < \mathbf{y}^T U(\mathcal{H})\mathbf{y} = \max(\mathbf{y}^T\mathcal{H}\mathbf{y})$ by Theorem 4.2.2 since $y_i = \pm 1\ \forall i$, so, by Lemma 4.2.1, U' is not an upper Hessian of $C(\mathcal{H})$. □

Corollary 4.3.1 shows $U(\mathcal{H})$ to be optimal in the very specific sense of minimizing trace. It can also be shown that $U(\mathcal{H})$ is optimal in a more general sense. It is *non-dominated*, that is, there is no other quadratic upper envelope which is everywhere less than or equal to the corresponding quadratic upper envelope of $U(\mathcal{H})$.

Corollary 4.3.2 *If U' is an upper Hessian of $C(\mathcal{H})$ and $\mathbf{y}^T U' \mathbf{y} \leq \mathbf{y}^T U(\mathcal{H}) \mathbf{y}$, $\forall \mathbf{y} \in \mathbb{R}^n$, where $U(\mathcal{H})$ is the associated upper Hessian of $\mathcal{H}$ obtained by Theorem 4.2.2, then $U' = U$.*

Proof: Suppose $\mathbf{y}^T U' \mathbf{y} \leq \mathbf{y}^T U(\mathcal{H}) \mathbf{y}$, $\forall \mathbf{y} \in \mathbb{R}^n$ but $U' \neq U(\mathcal{H})$. Thus $U' - U(\mathcal{H})$ is negative semi-definite and not equal to zero. Thus, the eigenvalues of $U' - U(\mathcal{H})$ are all less than or equal to zero, and at least one is less than zero. Hence, $\mathrm{tr}(U' - U(\mathcal{H})) < 0$, which implies that $\mathrm{tr}(U') < \mathrm{tr}(U(\mathcal{H}))$, and U' is not an upper Hessian by Corollary 4.3.1. □

The upper Hessian $U(\mathcal{H})$ is by no means the unique non-dominated upper Hessian. In general there are many non-dominated upper Hessians. For instance, an upper Hessian of minimum trace subject to the additional constraint of touching the piecewise quadratic upper envelope at an arbitrarily chosen differentiable point will be non-dominated by a similar argument.

4.4 Applications of Results

4.4.1 An Adaptive Second Derivative Algorithm

The results presented in this chapter were developed primarily for use in the adaptive second derivative global optimization algorithm built on the work of Baritompa and Cutler discussed in Chapter 5. This algorithm requires bounding Hessians to be found for f restricted to D, where D is a simplex or a box in R^n. Interval Hessians of f over D can be obtained by using generalised interval arithmetic. Theorem 4.2.2 then provides an $O(n^2)$ method of obtaining bounding Hessians. The optimality results, and the fact that $U(\mathcal{H})$ and $L(\mathcal{H})$ contain much of the original structure of the interval Hessian, suggest this method is likely to provide good pseudo-upper envelopes, at minimal cost $(O(n^2))$.

4.4.2 Reformulation of General Problems

Consider the very general problem:

$$\min_{\mathbf{x} \in D_0} F(\mathbf{x}), \text{ subject to } G_i(\mathbf{x}) \leq 0, \quad (1 \leq i \leq m), \tag{4.8}$$

where $F(\mathbf{x}), G_i(\mathbf{x}) \in C^2$ and D_0 is non-empty compact convex. Bounding Hessians can be useful on this problem in three ways: (a) to reformulate F and the G_i's as d.c. functions, (b) to reformulate the problem into standard GOP form, (c) to find convex under-estimators of the objective and constraint functions. (It is traditional to find global minima rather than maxima for problems in this form, so this convention shall be used for the remainder of this section.)

Adjiman et al. [1] suggest using a "Kharitonov-like" theorem to obtain lower bounds of the eigenvalues of the symmetric Hessian on subdomains from the interval Hessian. They do this for each variable of each (non-special) non-linear term in the objective and constraint functions. These bounds are then used to find convex under-estimators in a branch-and-bound framework to solve the global optimization problem (the αBB algorithm) [4]. The bounds could also be used to reformulate the problem as d.c. functions [69] or as GOP [39].

As an aside, Theorem 4.2.2 can be used, indirectly, to replace the "Kharitonov-like" theorem, by noting that the minimum eigenvalue of a symmetric lower Hessian is a lower bound on the eigenvalues of the Hessian at each point in the subdomain.

More importantly, Theorem 4.2.2 can be used directly, retaining more of the structure of the interval Hessian. The following proposition provides the key.

Proposition 4.4.1 *If L is a lower Hessian of f then $f(\mathbf{x}) - \frac{1}{2}\mathbf{x}^T L\mathbf{x}$ is convex.*

Proof: Convexity follows immediately from Theorem 4.2.1. □

Thus, we see that if L and L_i are lower Hessians of $F(\mathbf{x})$ and $G_i(\mathbf{x})$, for $1 \leq i \leq M$, respectively, then the three ways of using them, mentioned earlier, are:

(a) writing

$$\begin{aligned} F(\mathbf{x}) &= (F(\mathbf{x}) - \frac{1}{2}\mathbf{x}^T L\mathbf{x}) - (-\frac{1}{2}\mathbf{x}^T L\mathbf{x}), \\ G_i(\mathbf{x}) &= (G(\mathbf{x}) - \frac{1}{2}\mathbf{x}^T L_i\mathbf{x}) - (-\frac{1}{2}\mathbf{x}^T L_i\mathbf{x}), \quad (1 \leq i \leq m), \end{aligned}$$

gives an explicit d.c. representation of (4.8) if L and the L_i's are all negative semi-definite;

(b) letting

$$\begin{aligned} f(\mathbf{x},\mathbf{y}) &= F(\mathbf{x}) - \frac{1}{2}\mathbf{x}^T L\mathbf{x} + \frac{1}{2}\mathbf{x}^T L\mathbf{y}, \\ g_i(\mathbf{x},\mathbf{y}) &= G_i(\mathbf{x}) - \frac{1}{2}\mathbf{x}^T L_i\mathbf{x} + \frac{1}{2}\mathbf{x}^T L_i\mathbf{y}, \quad (1 \le i \le m), \text{ and} \\ h(\mathbf{x},\mathbf{y}) &= \mathbf{x} - \mathbf{y}, \end{aligned}$$

satisfies the GOP conditions if L and the L_i's are all negative semi-definite ($f(\cdot,\mathbf{y})$, $g_i(\cdot,\mathbf{y})$, $f(\mathbf{x},\cdot)$ and $g_i(\mathbf{x},\cdot)$ are differentiable convex functions for any fixed $\mathbf{y} \in D_0$ or $\mathbf{x} \in D_0$, and $h(\mathbf{x},\mathbf{y})$ is bilinear) and solving

$$\min_{\mathbf{x},\mathbf{y}\in D_0} f(\mathbf{x},\mathbf{y}) \quad \text{subject to} \quad \begin{aligned} g_i(\mathbf{x},\mathbf{y}) &\le 0, \quad (1 \le i \le m), \\ h(\mathbf{x},\mathbf{y}) &= 0, \end{aligned}$$

is equivalent to solving the original problem (4.8); and finally

(c) convex under-estimators of $F(\mathbf{x})$ and $G_i(\mathbf{x})$, $(1 \le i \le m)$, on D_0 are given by

$$\begin{aligned} f(\mathbf{x}) &= F(\mathbf{x}) - \frac{1}{2}\mathbf{x}^T L\mathbf{x} + c, \\ g_i(\mathbf{x}) &= G_i(\mathbf{x}) - \frac{1}{2}\mathbf{x}^T L\mathbf{x} + c_i, \quad (1 \le i \le m), \end{aligned}$$

where $c \le \frac{1}{2}\mathbf{x}^T L\mathbf{x}$ and $c_i \le \frac{1}{2}\mathbf{x}^T L_i\mathbf{x}$, $(1 \le i \le m)$, for all $\mathbf{x} \in D_0$, respectively.

Parts (a) and (b) require the lower Hessians to be negative semi-definite. If L is not negative semi-definite then the optimal, in the sense of minimizing the Frobenius norm, "negative semi-definite part of L", which remains a lower Hessian, can be found in $O(n^3)$ time using spectral decompositions. Alternatively, negative semi-definiteness can be replaced with the more stringent condition of diagonal dominance, which can be checked and, if necessary, enforced in $O(n^2)$ time. Part (c) does not require the lower Hessians to be negative semi-definite, but replacing L with the "negative semi-definite part of L" may lead to tighter lower bounds for $f(\mathbf{x})$.

Clearly, Proposition 4.4.1 and the above are true if the quadratic form $\frac{1}{2}\mathbf{x}^T L\mathbf{x}$ is replaced with any linear translate of it. For instance, if we are trying to find convex underestimators of $F(\mathbf{x})$ over D_0 then choosing a linear translate of $\mathbf{x}^T L\mathbf{x}$ which maximizes c gives better bounds for $F(\mathbf{x})$.

4.5 Summary and Conclusions

An $O(n^2)$ method to obtain non-dominated upper and lower Hessians of minimum trace from an interval Hessian has been provided. These bounding Hessians are not only useful for their originally intended purpose, the implementation of an adaptive second derivative covering method, but also for reformulation of very general global optimization problems into explicit forms required for a number of existing methods.

It remains to be seen how the performance of these methods would be affected by the adoption of this method compared to other ways to achieve these reformulations. For instance, it would be interesting to investigate a modified αBB algorithm [4] using the convex under-estimators obtained by the method described.

Chapter 5

Adaptive Second Derivative Algorithms

The purpose of this chapter is to bring together the theoretical results of the previous chapters into a global optimization algorithm. The algorithm presented is an adaptive second derivative branch and bound algorithm. In addition to finding the global maximum to within ϵ, the algorithm returns a superset of the global optimizers (an approximation to the set of global optimizers to within ϵ) which can be used to find associated global optimizers to within ϵ if desired.

In the design of the algorithm, a number of goals were kept in mind. Most importantly, it wa required be *theoretically sound*; that is, given any $\epsilon > 0$ it must halt after a finite time returning a global optimum to within ϵ. Secondary goals were to keep the amount of work required for each iteration tractable with respect to the problem size, to use as much information and structure of the problem as possible, to exploit the computer resources as fully as possible and to keep the description of the algorithm simple.

Before presenting the new algorithm, the Branch and Bound framework and the second derivative algorithms of Breiman, Cutler and Baritompa are discussed.

5.1 Branch and Bound Algorithms

Many algorithms, particularly deterministic algorithms, can be described in Horst and Tuy's [33] branch and bound framework (see also Zhang *et al.* [74] who extend

the framework to incorporate Wood's algorithm). Here a simplified framework, sufficient to describe the new algorithm, is presented.

In this simplified framework, it is assumed the domain of the objective function can be partitioned into a finite number of subdomains from $\mathcal{D}$, a class of subsets of $\mathbb{R}^n$. (In the more general framework the assumption is that a superset of the domain can be partitioned with sets from $\mathcal{D}$.) The simplified branch and bound framework is described as follows.

1. **(Initialisation)** Let $\mathcal{B} \subseteq \mathcal{D}$ be a finite partition of D_0. To each subdomain D in $\mathcal{B}$, associate an upper and lower bound of the global maximum of f restricted to D, denoted $M(D)$ and $m(D)$ respectively.

2. **(Choose)** Choose a subdomain $D \in \mathcal{B}$.

3. **(Subdivide)** Let $\mathcal{P} \subseteq \mathcal{D}$ be a partition of D.

4. **(Bound)** For each $D_i \in \mathcal{P}$, associate upper and lower bounds on the global maximum of f restricted to D_i, $M(D_i)$ and $m(D_i)$.

5. **(Update)** Remove D from $\mathcal{B}$ and add each subdomain $D_i \in \mathcal{P}$ to $\mathcal{B}$. Remove any subdomains $D \in \mathcal{B}$ with $M(D) < \max\{m(D_i) | D_i \in \mathcal{B}\}$. Such subdomains are called *fathomed*. (Also, any subdomain which, for any other reason, cannot contain the global maximum of f, is also called fathomed and can be removed from $\mathcal{B}$.)

6. **(Termination)** Stop if the *variation*,

$$\max\{M(D_i) | D_i \in \mathcal{B}\} - \max\{m(D_i) | D_i \in \mathcal{B}\}$$

   is less than ϵ. Otherwise, repeat from Step 2.

In order to describe a branch and bound algorithm one need only describe the *choosing rule* of Step 2, the *subdividing rule* of Step 3, and the *bounding rule* of Step 4. The following conditions are sufficient to assure the algorithm halts [33, 26].

B1 The associated upper bound on any subdomain D in $\mathcal{D}$ must tend to the maximum of f restricted to D as the diameter of D tends to zero.

B2 The diameter of any subdomain D in $\mathcal{D}$ must tend to zero as the number of subdivisions leading to it tends to infinity.

B3 The subdomain in $\mathcal{B}$ with the greatest associated upper bound must be chosen after a finite number of iterations.

After a branch and bound algorithm satisfying these conditions halts, the greatest associated lower bound over all subdomains in $\mathcal{B}$ is a global maximum to within ϵ. Also, the set of global maximizers to within ϵ is contained in $\bigcup \mathcal{B}$, called the *bracket.*

5.2 Algorithms of Baritompa and Cutler

The algorithm presented in this chapter uses ideas based on the second derivative covering methods of Baritompa and Cutler [6], which extend Breiman and Cutler's algorithm [8]. The following definitions are needed. Recall, a function $F : D_0 \to \mathbb{R}^n$ is an *upper envelope* of $f : D_0 \to \mathbb{R}^n$ if $F(\mathbf{x}) \geq f(\mathbf{x})$,for all $\mathbf{x} \in D_0$.

Definition 5.2.1 *A function is a* pseudo-upper envelope *of f if $F(\mathbf{x}^*) \geq f(\mathbf{x}^*)$, for all $\mathbf{x}^* \in X^*$.*

(The terminology "pseudo-upper envelope" is due to MacLagan *et al.* [40].)

Baritompa and Cutler present three related algorithms. The first uses an upper Hessian of f to generate upper envelopes. The second makes use of a lower Hessian to generate pseudo-upper envelopes. The final algorithm attempts to make use of both an upper and lower Hessian to generate improved pseudo-upper envelopes. An example is given showing this method, in its present form, is not valid in general. Restricted sufficient conditions under which it is valid are also given.

While the algorithms of Baritompa and Cutler can be described in the branch and bound framework, it is better to describe them using the following framework for covering algorithms.

1. **(Initialisation)** Let $k = 0$ and $\mathbf{x}_0 \in D_0$ be a *sample point.* Associate with $\mathbf{x}_0$ its function value $f(\mathbf{x}_0)$ and any other local information required.

2. **(Cover)** Using sample points and associated local information generate a (pseudo-)upper envelope F_k of f.

3. **(Termination)** Halt if the *variation*,

$$\max\{F_k(\mathbf{x}) : \mathbf{x} \in D_0\} - \max\{f(\mathbf{x}_i) | i = 0, \ldots, k\},$$

is less than ϵ.

4. **(Sample)** Let $\mathbf{x}_{k+1}$ be a maximizer of F_k and associate with it its function value and any other local information required.

5. **(Update)** Let $k = k + 1$ and repeat from 2.

In order to describe a covering algorithm one must specify how the (pseudo-) upper envelopes are generated and a method for determining its maximum and a maximizer. The latter is crucial, since this global optimization subproblem could potentially be as hard as the original problem. The following theorem gives sufficient conditions to assure the algorithm halts.

Theorem 5.2.1 *Suppose that $f : D_0 \to \mathbb{R}$ and $F_k(\mathbf{x})$, for $k \in \{0, 1, 2, \ldots\}$, are the (pseudo-)upper envelopes of f generated by a covering algorithm after k iterations with $\epsilon = 0$ and satisfying:*

C1 *The (pseudo-)upper envelopes are equal to the function value at the sample points, that is, $F_k(\mathbf{x}_i) = f(\mathbf{x}_i)$, $\forall k, \forall i = \{0, \ldots, k\}$.*

C2 *The (pseudo-) upper envelopes are all uniformly continuous with respect to k as follows. Given any $\epsilon > 0$, $\delta(\epsilon) > 0$ can be found so that $\|\mathbf{x} - \mathbf{y}\| < \delta(\epsilon)$ implies $|F_k(\mathbf{x}) - F_k(\mathbf{y})| < \epsilon$, $\forall k, \forall \mathbf{x}, \mathbf{y} \in D_0$.*

Then the algorithm halts for any $\epsilon > 0$.

Proof: Fix $\epsilon > 0$. Let $\delta(\epsilon)$ be as specified in (C2). For any k, $\mathbf{x}_{k+1}$ cannot be within $\delta(\epsilon)$ of any point $\mathbf{x}_i$, $i = \{0, \ldots, k\}$ because this would imply that the variation was less than ϵ and the algorithm would have halted. But D_0 is compact, so only a finite number of points all separated by at least $\delta(\epsilon)$ can be in D_0. Therefore the algorithm halts. □

When a covering algorithm terminates, the maximum function value of all the sampled points, $\underline{y}^*$, is a global maximum to within ϵ. Also, the set of global maximizers to within ϵ is contained in the set $\{\mathbf{x} | F(\mathbf{x}) \geq \underline{y}^*\}$. This set may not be easy to determine.

5.2.1 Using Upper Hessians

Suppose U is a positive semi-definite upper Hessian of $f : D_0 \to \mathbb{R}$, where $D_0 \subseteq \mathbb{R}^n$ is a polytope. That is,

$$f(\mathbf{x}) \leq f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x} - \mathbf{a}) + (\mathbf{x} - \mathbf{a})^T U(\mathbf{x} - \mathbf{a})/2 \tag{5.1}$$

for all $\mathbf{x}$ and $\mathbf{a}$ in D_0. It follows that if $(\mathbf{x}_0, \ldots, \mathbf{x}_k)$ are sample points in D_0

$$F_k(\mathbf{x}) = \min_{i=0,\ldots,k} \left(f(\mathbf{x}_i) + \nabla f(\mathbf{x}_i)^T(\mathbf{x} - \mathbf{x}_i) + (\mathbf{x} - \mathbf{x}_i)^T U(\mathbf{x} - \mathbf{x}_i)/2 \right)$$

is an upper-envelope of f. Condition (C1) is clearly satisfied. Since D_0 is compact, ∇f is bounded on D_0. Thus, for all k, F_k is Lipschitz continous with some Lipschitz constant L independant of k and Condition (C2) follows.

It remains to describe the method of obtaining the maximum and maximizers of the upper envelopes. In order to achieve this, a graph $G = (V, E)$ is maintained. The graph is initialised with the V and E being the vertices and edges of D_0 respectively.

After k iterations, the upper envelope is a piecewise quadratic function with each quadratic piece a defined on a polytope containing a sample point (see Figure 5.1). The vertices and edges of the graph are the vertices and edges of the polytopes. Since on each polytope, the upper envelope is a positive semi-definite quadratic, the maximum is achieved at a vertex.

As pointed out by Breiman and Cutler [8], it is not necessary to recompute all of G at every iteration. The graph can be updated efficiently after sampling a new point $\mathbf{x}_k$. This is achieved by identifying "dead" vertices, which are removed, and inserting new vertices. A vertex v is "dead" if $F_k(v) > f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(v - \mathbf{x}_k) + (v - \mathbf{x}_k)^T U(v - \mathbf{x}_k)/2$. Not all vertices need to be checked since the dead vertices are on a connected component of the graph. The new vertices are on edges between dead vertices and live vertices. They can be found by solving certain linear systems.

5.2.2 Using Lower Hessians

Suppose that L is a negative semi-definite lower Hessian of f, $\mathbf{x}^*$ is a global maximizer of f and $\nabla f(\mathbf{x}^*) = 0$. That is

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) + (\mathbf{x} - \mathbf{x}^*)^T L(\mathbf{x} - \mathbf{x}^*)/2 \tag{5.2}$$

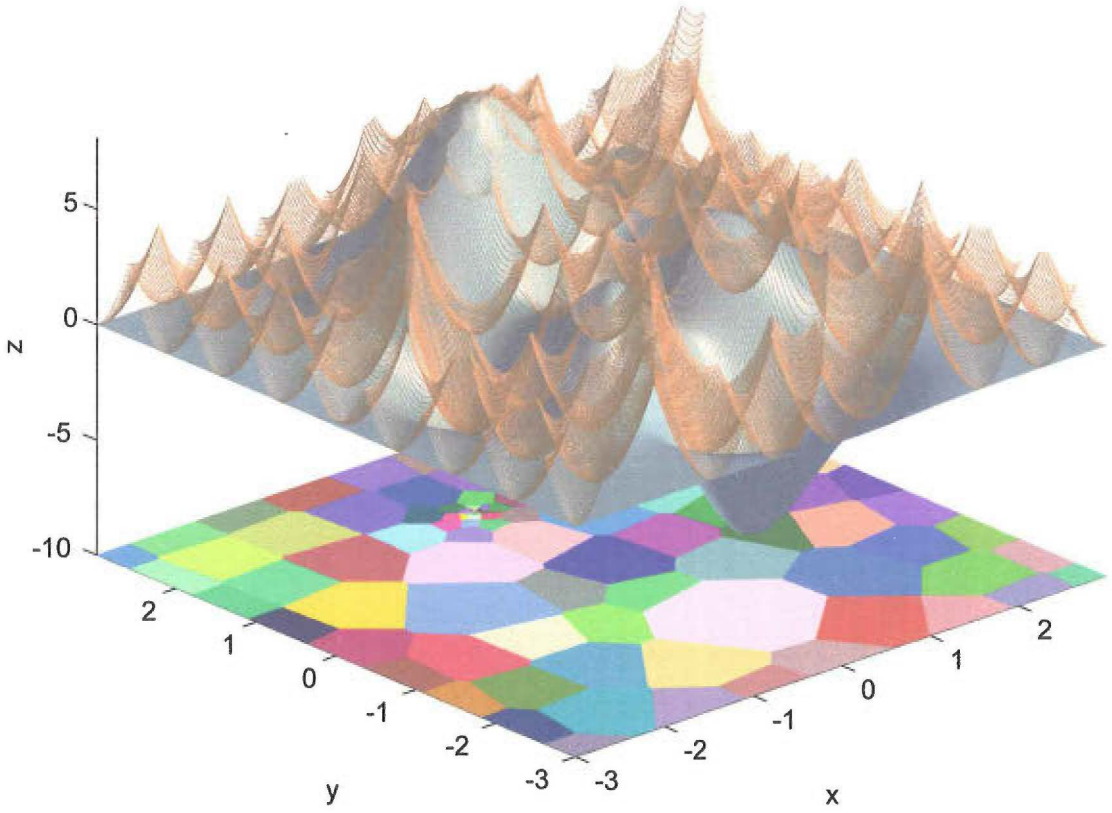

Figure 5.1: The Breiman and Cutler upper envelope (copper wire-frame) obtained by using the upper Hessian $U = 35I$ on top of the Peaks function (grey surface) after 100 iterations. Underneath are the projections onto the domain of the quadratics pieces that make up the envelope (randomly coloured). Each quadratic piece is defined on a polytope, the edges and vertices of these forming a graph.

for all $\mathbf{x} \in D_0$. (In fact, this is the only required condition on L, but no general method is known finding such an L that is not also a lower Hessian.) Rearranging (5.2) gives

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) - (\mathbf{x}^* - \mathbf{x})^T L(\mathbf{x}^* - \mathbf{x})/2$$

for all $\mathbf{x} \in D_0$. (Note that this inequality holds for any point with zero gradient.) It follows that if $(\mathbf{x}_0, \ldots, \mathbf{x}_k)$ are sample points in D_0,

$$F_k(\mathbf{x}) = \min_{i=0,\ldots,k} \left(f(\mathbf{x}_i) - (\mathbf{x}_i - \mathbf{x})^T L(\mathbf{x} - \mathbf{x}_i)/2 \right)$$

is a pseudo-upper envelope of f if the gradient at the global maximizers is zero (see Figure 5.2). This assumption is valid for C^2 functions with the global optimizers in the interior. The algorithm is implemented exactly as above, with U replaced by $-L$ and the gradient at $\mathbf{x}_i$ replaced with zero.

5.2.3 Using both Upper and Lower Hessians

Suppose that U and L are (semi-definite) upper and lower Hessians of f as above. Then combining Inequalities (5.1) and (5.2) gives

$$f(\mathbf{x}^*) \leq \min_{\mathbf{x} \in D_0} \left(f(\mathbf{a}) + \nabla f(\mathbf{a})^T (\mathbf{x} - \mathbf{a}) + (\mathbf{x} - \mathbf{a})^T U(\mathbf{x} - \mathbf{a})/2 - (\mathbf{x} - \mathbf{x}^*)^T L(\mathbf{x} - \mathbf{x}^*)/2 \right) \tag{5.3}$$

for all $\mathbf{a} \in D_0$ and $\mathbf{x}^* \in X^*$ where $\nabla f(\mathbf{x}^*) = 0$. Baritompa and Cutler find $\mathbf{x}$ so that the gradient with respect to $\mathbf{x}$ of the right-hand side of (5.3) is zero in the case of U and $-L$ positive definite. They show that this can be implemented by replacing U with $-UL(U - L)^{-1}$ (assuming U and L commute), and for $i = 0, \ldots, k$, replacing $\mathbf{x}_i$ with $\mathbf{x}_i - U^{-1}\nabla f(\mathbf{x}_i)$, $f(\mathbf{x}_i)$ with $f(\mathbf{x}_i) - \nabla f(\mathbf{x})^T U^{-1} \nabla f(\mathbf{x}_i)$ and the gradient at $\mathbf{x}_i$ with zero in the implementation of the first algorithm.

Unfortunately, this fails to take account of the boundary of D_0. The minimum of the right-hand side of (5.3) may not be where the gradient with respect to $\mathbf{x}$ is zero. The following result provides a sufficient condition for Baritompa and Cutler's acceleration to be valid. For some functions this condition is easily verified, though it may be hard to verify in general.

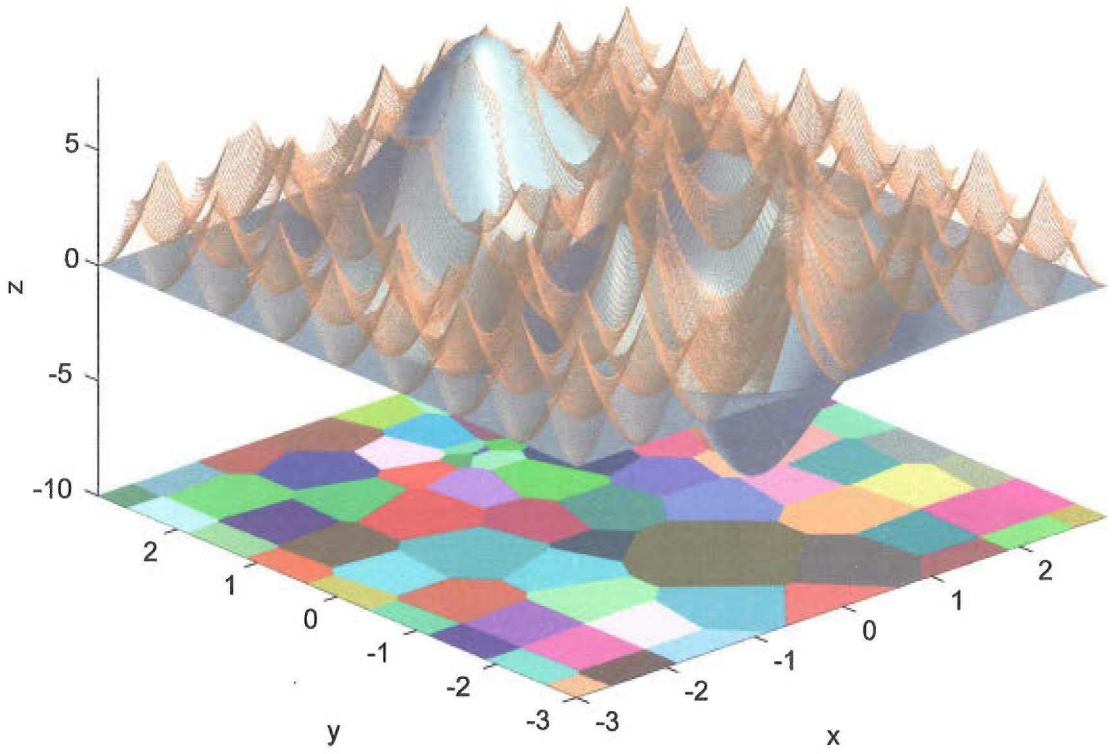

Figure 5.2: The Breiman and Cutler pseudo-upper envelope (copper-wire frame) obtained from $L = -35I$ on top of the Peaks function (grey surface) after 100 iterations. While the pseudo-upper envelope is not everywhere above the function, it is greater than the global maximum at the global maximizer. The projections onto the domain of quadratic pieces that make up the pseudo-upper envelope are polytopes as before.

Lemma 5.2.1 *If $f : D_0 \to \mathbb{R}$ has an extension $g : \mathbb{R}^n \to \mathbb{R}$ such that U and L are upper and lower Hessians of g then*

$$f(\mathbf{x}^*) \leq \min_{\mathbf{x}\in\mathbb{R}^n} \left(f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x}-\mathbf{a}) + (\mathbf{x}-\mathbf{a})^T U(\mathbf{x}-\mathbf{a})/2 - (\mathbf{x}-\mathbf{x}^*)^T L(\mathbf{x}-\mathbf{x}^*)/2 \right), \tag{5.4}$$

where $\mathbf{a} \in D_0$, $\mathbf{x}^* \in \mathbf{X}^*$ and $\nabla f(\mathbf{x}^*) = 0$.

Proof: Since $\nabla f(\mathbf{x}^*) = \nabla g(\mathbf{x}^*) = 0$ and L is a lower Hessian of g,

$$f(\mathbf{x}^*) \leq g(\mathbf{x}) - (\mathbf{x}-\mathbf{x}^*)^T L(\mathbf{x}-\mathbf{x}^*)/2,$$

for all $\mathbf{x} \in \mathbb{R}^n$. Also, since U is an upper Hessian of g,

$$g(\mathbf{x}) \leq g(\mathbf{a}) + \nabla g(\mathbf{a})^T(\mathbf{x}-\mathbf{a}) + (\mathbf{x}-\mathbf{a})^T U(\mathbf{x}-\mathbf{a})/2,$$

for all $\mathbf{x}, \mathbf{a} \in \mathbb{R}^n$. Combining these inequalities gives

$$f(\mathbf{x}^*) \leq \min_{\mathbf{x}\in\mathbb{R}^n} \left(g(\mathbf{a}) + \nabla g(\mathbf{a})^T(\mathbf{x}-\mathbf{a}) + (\mathbf{x}-\mathbf{a})^T U(\mathbf{x}-\mathbf{a})/2 - (\mathbf{x}-\mathbf{x}^*)^T L(\mathbf{x}-\mathbf{x}^*)/2 \right),$$

for all $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{a} \in D_0$. □

Thus, if f has an extension then Baritompa and Cutler's accelerations are valid. When f is univariate and defined on an interval there is always such an extension. If f is assumed C^2 then this can be easily verified using Theorem 4.2.1. The following result is more general.

Theorem 5.2.2 *Let $f : [a,b] \to \mathbb{R}$ and suppose that U and L are upper and lower Hessians of f. Then there exists $g : \mathbb{R} \to \mathbb{R}$ such that $g(x) = f(x)$, $\forall x \in [a,b]$ and U and L are upper and lower Hessians of g.*

Proof: Define $g : \mathbb{R} \to \mathbb{R}$ by

$$g(x) = \begin{cases} f(a) + f'(a)(x-a) & \text{if } x < a \\ f(x) & \text{if } x \in [a,b] \\ f(b) + f'(b)(x-b) & \text{if } x > b. \end{cases}$$

Clearly g extends f. Let $q_c(x) = g(c) + g'(c)(x-c) + U(x-c)^2/2$ for a fixed c. We wish show that U is an upper Hessian of g, that is $q_c(x) \geq g(x)$, $\forall x \in \mathbb{R}$, $\forall c \in \mathbb{R}$.

Suppose that $c \in [a,b]$. Then $q_c(x) \geq g(x) = f(x)$, $\forall x \in [a,b]$ since U is an upper Hessian of f. Also, $q_c(x) \geq q_b(x) \geq g(x)$, $\forall x > b$ since $q_c(x) - q_b(x)$ is linear,

less than or equal to zero at c and greater than or equal to zero at b. Similarly, $q_c(x) \geq q_a(x) \geq g(x)$, $\forall x < a$.

Now, suppose that $c < a$. Then $q_c(x) \geq g(x)$, $\forall x < a$ since $g(x)$, $\forall x < a$ is the tangent line of $q_c(x)$ at c. Also, $q_c(x) \geq q_a(x) \geq g(x)$, $\forall x \in [a, b]$ and $q_c(x) \geq q_a(x) \geq q_b(x) \geq g(x)$, $\forall x > b$. The case for $c > b$ is similar.

Therefore U is an upper Hessian of g, an extension of f. Similarly, L is a lower Hessian of g. □

Thus Baritompa and Cutler's acceleration is valid for univariate functions. Unfortunately, this result cannot be extended to higher dimensions, as an such an extension of f does not always exist. This is demonstrated by the following (non-trival!) 2-dimensional example.

Counter Example 5.2.1 *Let*

$$f(x,y) = \frac{1}{2}x(64 - 129x) + 16x\sqrt{1+16x^2} + (4+8y)\sinh^{-1}(4x) - \frac{1}{2}y^2\left(9 + 32\frac{x}{\sqrt{1+16x^2}}\right).$$

A box containing the origin and the point $(1,0)$ *can be found such that*

$$U = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \text{ and } L = \begin{bmatrix} -258 & 0 \\ 0 & -18 \end{bmatrix}$$

are upper and lower Hessians of f *restricted to this box. The global maximum of zero occurs at the origin. However, if* $\mathbf{a} = (1,0)$ *and* $\mathbf{x}^* = (0,0)$,

$$\min_{\mathbf{x}\in\mathbb{R}^n} \left(f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x}-\mathbf{a}) + (\mathbf{x}-\mathbf{a})^T U(\mathbf{x}-\mathbf{a})/2 - (\mathbf{x}-\mathbf{x}^*)^T L(\mathbf{x}-\mathbf{x}^*)/2\right) < 0. \tag{5.5}$$

Thus the acceleration is not valid.

Proof: Theorem 4.2.1 is used to show that U and L are valid upper and lower Hessians on some box containing the points $(0,0)$ and $(1,0)$. The partial derivatives of f, after simplification, are

$$f_x(x,y) = 32 - 129x + \frac{512x^2+32}{\sqrt{1+16x^2}} + 16\frac{y(2+32x^2-y)}{(1+16x^2)^{3/2}}$$

and

$$f_y(x,y) = 8\sinh^{-1}(4x) - y\left(9 + 32\frac{x}{\sqrt{1+16x^2}}\right).$$

The second partial derivatives are

$$f_{xx}(x,y) = -129 + \frac{512x}{\sqrt{1+16x^2}} - 256\frac{xy(2+32x^2-3y)}{(1+16x^2)^{5/2}}$$

$$f_{xy}(x,y) = 32\frac{1}{\sqrt{1+16x^2}} - y\left(32\frac{1}{\sqrt{1+16x^2}} - 512\frac{x^2}{(1+16x^2)^{3/2}}\right)$$

and

$$f_{yy}(x,y) = -9 - 32\frac{x}{\sqrt{1+16x^2}}.$$

Thus

$$\begin{aligned}\det(U - H(x,0)) &= \begin{vmatrix} 130 - 128\sin(\theta) & -8\cos(\theta) \\ -8\cos(\theta) & 10 + 8\sin(\theta) \end{vmatrix} \\ &= 276 - 240\sin(\theta) + 960\cos^2(\theta) > 0\end{aligned}$$

and

$$\begin{aligned}\det(H(x,0) - L) &= \begin{vmatrix} 129 + 128\sin(\theta) & 8\cos(\theta) \\ 8\cos(\theta) & 9 - 8\sin(\theta) \end{vmatrix} \\ &= 137 + 120\sin(\theta) + 960\cos^2(\theta) > 0.\end{aligned}$$

where $\theta = \tan^{-1}(4x)$. Since the determinants and diagonal elements of $U - H(x,0)$ and $H(x,0) - L$ are positive, $U - H(x,0)$ and $H(x,0) - L$ are positive definite. Since the second derivatives are continuous, it follows that $U - H(x,y)$ and $H(x,y) - L$ are positive definite for all (x,y) in some box D_0 containing the points $(0,0)$ and $(1,0)$. By Theorem 4.2.1, U and L are valid upper and lower Hessians of f restricted to D_0. Further, D_0 can be chosen so that $H(x,y)$ is positive definite for all $(x,y) \in D_0$ (since $H(x,0)$ is positive definite for all x). Since $f_x(0,0) = f_y(0,0) = 0$, the global maximizer is at the origin. By evaluating at the origin we find the global maximum is zero.

Finally, using the replacement values given by Baritompa and Cutler,

$$\begin{aligned} U^a &= -UL(U-L) = \begin{bmatrix} 258/259 & 0 \\ 0 & 18/19 \end{bmatrix}, \\ \mathbf{x}^a &= (1,0) - U^{-1}\nabla f(1,0) \\ &= \left(98 - 32\sqrt{17}, -8\sinh^{-1}(4)\right)^T, \text{ and} \\ f^a &= f(1,0) - \nabla f(1,0)^T L^{-1}\nabla f(1,0)/2 \\ &= 3120\sqrt{(17)} + 4\sinh^{-1}(4) - 32(\sinh^{-1}(4))^2 - 13441, \end{aligned}$$

and letting $\mathbf{a} = (1,0)$ and $\mathbf{x}^* = (0,0)$ we obtain

$$\begin{aligned} & \min_{\mathbf{x}\in\mathbb{R}^n} \left(f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x}-\mathbf{a}) + (\mathbf{x}-\mathbf{a})^T L(\mathbf{x}-\mathbf{a})/2 - (\mathbf{x}-\mathbf{x}^*)^T U(\mathbf{x}-\mathbf{x}^*)/2\right) \\ = \; & f_a + (\mathbf{x}^* - \mathbf{x}^a)^T U^a(\mathbf{x}^* - \mathbf{x}^a) \\ = \; & -\frac{144}{37}\sqrt{(17)} + 4\sinh^{-1}(4) + \frac{3329}{259} - \frac{32}{19}(\sinh^{-1}(4))^2 \\ < \; & 0. \end{aligned}$$

Hence, the acceleration is not valid. □

When no extension of f exists (as in the above example), or if it is not known whether such an extension exists, it is not clear how both upper and lower Hessians of the objective function can be used effectively. One possibility is to find the minimum of the right-hand side of Inequality (5.3) computationally rather than analytically. However, the pseudo-upper envelope so obtained may not have the structure that makes it possible to find the maximum and maximizers. Further investigation into this area is required.

5.3 The New Algorithm

Breiman and Cutler point out two failings of second derivative covering methods. Firstly, no method for finding good or even valid bounding Hessians is provided. The method described in Chapter 4 addresses this problem, although the bounding Hessians so obtained may not be very tight. Secondly, the bounding Hessians "may differ drastically over different subregions of D_0". Since the bounding Hessian is pre-computed there is no scope to adapt to a better bounding Hessian when more about the function is known (in particular, when certain regions of the domain are now known not to contain the global optimum).

The algorithm presented here addresses this problem. The domain is broken into subdomains, as in a branch and bound algorithm, and associated with each of these a valid interval Hessian is found. Bounding rules using these interval Hessians, or bounding Hessians obtained from them, are used to bound the function. The bounding rules found using bounding Hessians are based on the (pseudo-)upper envelopes of Baritompa and Cutler's algorithms. A valid method for using both upper and lower Hessians is also presented. Thus the new algorithm can be described as an adaptive second derivative branch and bound algorithm.

To describe the new algorithm it is sufficient to describe the bounding, subdividing and choosing rules. A discussion of these is presented first, followed by a summary.

5.3.1 The Bounding Rules

A bounding rule is defined on $\mathcal{D}$, a class of subsets of $\mathbb{R}^n$. If just interval arithmetic is used to obtain bounds on subdomains, then we are restricted to $\mathcal{D}$ being the set of all boxes. However, if generalised interval arithmetic is used then $\mathcal{D}$ can be any class for which a suitable set of base subexpressions, and inclusion functions for these over $\mathcal{D}$, can be found. The choice of $\mathcal{D}$ should be considered in parallel with the choice of bounding rule, and take into account such things as the structure of the domain and the structure of the expressions to be bounded.

For this discussion let us restrict our attention to $\mathcal{D}$ being either the set of all boxes in D_0 or the set of all simplexes in D_0. If D_0 is a finite union of boxes then the set of all boxes is the natural choice for $\mathcal{D}$. However, such a domain could be partitioned into a finite (possibly large) number of simplexes and $\mathcal{D}$ chosen to be the set of all simplexes. This may be worthwhile, for instance, if a very efficient generalised interval arithmetic can be used on simplexes but not boxes. On the other hand, if the domain is a finite union of simplexes, for instance if it is a general polytope, then the set of all simplexes makes a natural choice. Other classes worth considering are the set of all polytopes, the set of all sets bounded by linear constraints and the set of all balls.

First some potential bounding rules based on interval or bounding Hessians will be discussed. Given $D \in \mathcal{D}$, interval Hessians of f restricted to D, denoted $\mathcal{H}(D)$, are obtained by using generalised interval arithmetic, described in Chapter 3, to

find bounds on the partial second derivatives of f restricted to D. Upper and lower Hessians of f restricted to D, denoted $U(D)$ and $L(D)$ respectively, are obtained from $\mathcal{H}(D)$ by the method of Theorem 4.2.2 in Chapter 4.

Second, in order to compare prospective rules the convergence rate of the bounding rule is considered. The *order of convergence of a bounding rule* is the order with which the difference between the upper and lower bounds in the global maximum of f restricted to D approaches zero as the diameter of D tends to zero. For the complexity results it is assumed that the width of the entries of $\mathcal{H}(D)$ are $O(\text{diam}(D))$. This is the case, for instance, if all the subexpressions of the second partial derivatives are Lipschitz continuous on D_0.

Finally it is assumed that X^* is in the interior of D_0. For some of the bounding rules this assumption is unnecessary.

5.3.1.1 Using Interval Hessians

The only adaptive second derivative bounding rule in the literature is to use the second order Taylor form to give,

$$\max_{\mathbf{x}\in\mathbf{X}} f(\mathbf{x}) \in \left[f(\mathbf{a}), \max\left(f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{X}-\mathbf{a}) + (\mathbf{X}-\mathbf{a})^T\mathcal{H}(\mathbf{X})(\mathbf{X}-\mathbf{a})/2)\right)\right],$$

where $\mathbf{a}$ is the midpoint of the box $\mathbf{X}$. (Recall, the "max" of the right-hand endpoint is the maximum of the interval given by the expression.) While the excess width converges to zero quadratically [57], the bounding rule has only linear convergence in general (consider $f(x) = x$).

In addition, a limited but fast convexity test can be performed. Since the Hessian at a global maximizer is negative semi-definite, if the diagonal elements of $\mathcal{H}(\mathbf{X})$ are all greater than zero (that is, the minima of the intervals are all greater than zero) then $\mathbf{X}$ can be marked as fathomed.

However, the following *modified second order Taylor form* can be used instead (cf. [5]). Since the gradient at a (interior) global maximizer is zero, the second order Taylor form about $\mathbf{x}^* \in \mathbf{X}^*$ can be rearranged to give

$$\max_{\mathbf{x}\in\mathbf{X}^*\cap\mathbf{X}} f(\mathbf{x}) \in \left[f(\mathbf{a}), \max\left(f(\mathbf{a}) - (\mathbf{X}-\mathbf{a})^T\mathcal{H}(\mathbf{X})(\mathbf{X}-\mathbf{a})/2\right)\right],$$

where $\mathbf{a}$ is the midpoint of $\mathbf{X}$. To the author's knowledge this interval inclusion has not be observed before. It is faster to evaluate since the gradient at $\mathbf{a}$ is not needed

(although $O(n^2)$ operations are still required) and, since $(\mathbf{X}-\mathbf{a})$ is symmetric, always tighter unless $\nabla f(\mathbf{a}) = 0$. Furthermore, it has a quadratic rate of convergence. (This follows from the fact that the widths of $\mathcal{H}(\mathbf{X})$ are bounded, the width of $(\mathbf{X} - \mathbf{a})$ is linearly convergent and the width of products of two intervals is Lipschitz. To see the bound is tight consider $f(x) = -x^2$.)

5.3.1.2 Using Upper Hessians

The upper Hessian of f restricted to D_0 can be used to obtain bounds on the global maximum of f restricted to D in a similar fashion to the first of Baritompa and Cutler's algorithms discussed above. From Inequality 5.1 we have

$$\max_{\mathbf{x}\in D} f(\mathbf{x}) \in \left[f(\mathbf{a}), \max_{\mathbf{x}\in D} \left(f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x}-\mathbf{a}) + (\mathbf{x}-\mathbf{a})^T U(D)(\mathbf{x}-\mathbf{a})/2 \right) \right].$$

The upper bound can be found in $O(n^3)$ time if D is a simplex and $U(D)$ is positive semi-definite by evaluating the quadratic at the vertices. In general, if $U(D)$ is indefinite or D is a box then the upper bound cannot be found in polynomial time [70].

The upper Hessian $U(D)$ can be replaced by $U_+(D)$, a positive semi-definite upper Hessian of $f|_D$ found by using spectral decompositions. This however degrades the bound and, more importantly, leads to only a linear order of convergence (since a concave function is bounded above by, at best, a linear function).

5.3.1.3 Using Lower Hessians

Alternatively, the lower Hessian of f restricted to D could be used as in the second algorithm of Breiman and Cutler discussed above. It follows that

$$\max_{\mathbf{x}\in D\cap X^*} f(\mathbf{x}) \in \left[f(\mathbf{a}), \max_{\mathbf{x}\in D} (f(\mathbf{a}) - (\mathbf{x}-\mathbf{a})L(D)(\mathbf{x}-\mathbf{a})/2) \right].$$

A full convexity test can be performed—if $L(D)$ is not negative semi-definite then D can be marked as fathomed since it cannot contain a global maximizer with negative semi-definite Hessian. Thus, we can assume that $L(D)$ is negative semi-definite for the remainder. The upper bound can be found in $O(n^3)$ on simplexes by evaluating at the vertices. In general, the upper bound cannot be found in polynomial time on boxes. This bounding rule is quadratically convergent (the proof follows by similar considerations to the modified Taylor form above).

5.3.1.4 Using both Upper and Lower Hessians

It is somewhat surprising at first that the above reasonably intuitive (and polynomial time) bounding rules using adaptive second derivatives lead to only quadratic convergence at the most. After all, the second order Taylor expansion converges to the function cubicly and can be evaluated in $O(n^3)$ time (but cannot be used because it does not provide upper bounds). The following bounding rule achieves a cubic order of convergence with $O(n^3)$ work, but is less natural than the above.

Observe that $L(D)$ can be used to find a *lower* bound on all function values in D, and hence

$$\max_{\mathbf{x}\in D} f(\mathbf{x}) \geq \max_{\mathbf{x}\in D} \left(f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x}-\mathbf{a}) + (\mathbf{x}-\mathbf{a})^T L(D)(\mathbf{x}-\mathbf{a})/2 \right). \tag{5.6}$$

Since D can be marked as fathomed if $L(D)$ is not negative semi-definite (assuming the global maximizers are in the interior of D_0) we need only consider the case when $L(D)$ is negative semi-definite. The problem of maximising a negative semi-definite quadratic over a box or a simplex can be achieved in $O(n^3 L)$ time, where L is the problem length, by interior point methods [20].

To obtain an upper bound that will lead to cubic convergence, note that the difference between the upper and lower quadratic envelopes of f corresponding to $U(D)$ and $L(D)$ respectively is $(\mathbf{x}-\mathbf{a})(U(D)-L(D))(\mathbf{x}-\mathbf{a})/2$. It follows that

$$\begin{aligned}\max_{\mathbf{x}\in D} f(\mathbf{x}) \leq \max_{\mathbf{x}\in D} &\left(f(\mathbf{a}) + \nabla f(\mathbf{a})^T(\mathbf{x}-\mathbf{a}) + (\mathbf{x}-\mathbf{a})^T L(D)(\mathbf{x}-\mathbf{a})/2 \right) \\ &+ \max_{\mathbf{x}\in D} (\mathbf{x}-\mathbf{a})^T(U(D)-L(D))(\mathbf{x}-\mathbf{a})/2\end{aligned}$$

Note that $U(D)-L(D)$ is always positive semi-definite and, in fact, is always diagonal if obtained by the method of Theorem 4.2.2. Therefore, $\max_{\mathbf{x}\in D}(\mathbf{x}-\mathbf{a})(U(D)-L(D))(\mathbf{x}-\mathbf{a})$ can be obtained in $O(n^2)$ time on simplexes by evaluating at the vertices and $O(n)$ time on boxes by evaluating at only one vertex (by a symmetry argument).

This bound has a cubic convergence rate as shown by the following Theorem.

Theorem 5.3.1 *Suppose* $w(\mathcal{H}(D)) = O(\mathrm{diam}(D))$ *and* $U(D)$ *and* $L(D)$ *are upper and lower Hessians of* f *over* D *obtained from* $\mathcal{H}(D)$ *by the method described in Chapter 4. If* $\mathbf{a} \in D$ *then* $\max_{\mathbf{x}\in D}(\mathbf{x}-\mathbf{a})^T(U(D)-L(D))(\mathbf{x}-\mathbf{a})$, *is* $O(\mathrm{diam}(D)^3)$.

Proof: Fix f. Note that if $\mathbf{a}$ and $\mathbf{x}$ are in D then $(\mathbf{x}_i - \mathbf{a}_i)$ is $O(\text{diam}(D))$. Also, the diagonal elements of $U(D) - L(D)$, $U_{ii} - L_{ii} = \sum_{j=1}^{n} w(H_{ij}(D)) + w(H_{ji}(D)) = O(\text{diam}(D))$. Finally, $(\mathbf{x}-\mathbf{a})^T(U(D)-L(D))(\mathbf{x}-\mathbf{a}) = \sum_{i=1}^{n}(\mathbf{x}_i-\mathbf{a}_i)(U_{ii}-L_{ii})(\mathbf{x}_i - \mathbf{a}_i) = \sum_{i=1}^{n} O(\text{diam}(D)^3) = O(\text{diam}(D)^3)$. □

To the author's knowledge, this is the first time a cubicly convergent bounding rule has been used for a global optimization algorithm.

5.3.1.5 Other bounding rules

Computation experience reveals that it is very inefficient to use only adaptive second derivative bounding rules. Such bounding rules can lead to comparatively good bounds for subdomains of small diameter, but tend to give very poor bounds for subdomains of large diameter. For this reason they should be combined with bounding rules that give better bounds for subdomains of large diameter. In particular, if $\mathcal{D}$ is the set of all boxes then the natural inclusion function (or an improvement on it using generalised interval arithmetic), a monotonicity test (checking the inclusions of the partial derivatives of f restricted to $\mathbf{X}$ all contain zero) and a first order Taylor expansion can be used. Similar bounding rules could be use on other classes of $\mathcal{D}$ using generalised interval arithmetic.

If automatic differentiation is used to obtain the interval Hessian, then the natural inclusion interval and an interval gradient are obtained at no extra cost.

5.3.2 The Subdividing Rule

A simple subdivision rule of trisecting along the longest edge is used. Clearly such a bounding rule satisfies condition (B2). Trisection is chosen over bisection because it saves one function (and gradient) evaluation per iteration. See Figure 5.3.

5.3.3 The Choosing Rule

Two commonly used choosing rules are *best-first*, choosing the subdomain with the largest upper bound, and *width-first*, choosing the oldest simplex. A best-first choosing rule tends to work better in practice although the overheads are slightly higher than width first. A width first choosing rule also guarantees the union of subdomains tends to X^* in the limit. Both of these choosing rules can require a lot of

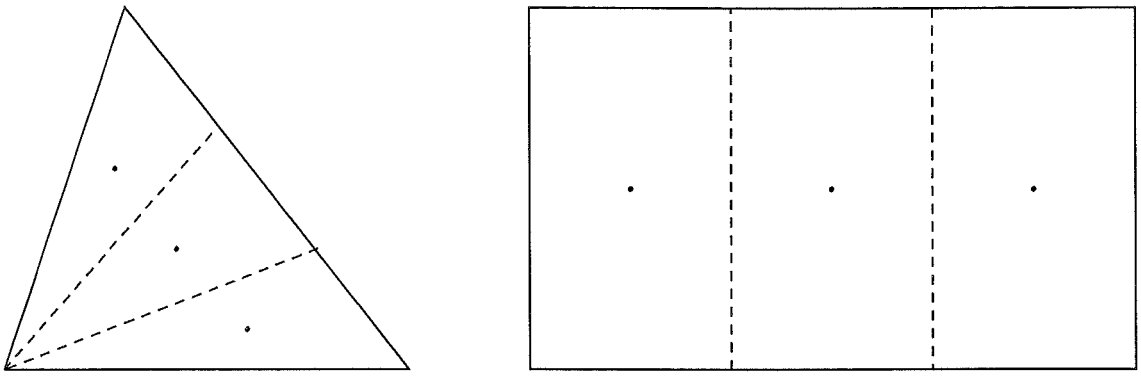

Figure 5.3: The simplex or box is trisected along the longest edge. The function and gradient need only be evaluated at the two new midpoints (cenroids for the simplex).

memory storage.

Computation experience reveals that this can be a limiting factor for large problems. On modern computer architectures with virtual memory, the algorithm can slow by several orders of magnitude when real memory is full.

The author and Murray investigated a *depth-first* choosing rule, that is, choosing the youngest subdomain, for a Piyavskii-Shubert like algorithm in [46]. Depth-first choosing rules have also been investigated by other authors [47, 36, 58]. A depth-first algorithm requires only about the logarithm of the memory requirement of a best- or width-first and so is very memory efficient. The down side is that a "bad" choice early on in the algorithm leads to extensive searching of a region with no global maximizers. Only when the global maximum to within ϵ of f over that region is found will the algorithm start subdividing the "good" region.

With these considerations in mind a hybrid best-first/depth-first choosing rule was chosen. The algorithm uses a best-first choosing rule as long as the (real) memory permits. Whenever the memory is almost full, it uses a depth-first strategy to search the best (and most promising) domain. Subdomains where the difference between the upper and lower bound is less than ϵ are not subdivided further and are stored.

5.4 Summary

The following summarises the bounding, subdividing and choosing rules of a global optimization algorithm to solve essentially unconstrained problems defined on (a union of) boxes.

- **Bounding Rule** Let $\underline{f}^*$ be an underestimate of the global maximum of f and $\mathbf{X}$ be a box with mid-point $\mathbf{a}$. The following bounding rule is used. It returns $M(\mathbf{X})$ the associated upper bound of the global maximum of f restricted to $\mathbf{X}$ and updates $\underline{f}^*$.

  1. (**Evaluation**) Evaluate inclusions of the function, gradient and Hessian over $\mathbf{X}$, $f(\mathbf{X})$, $\nabla f(\mathbf{X})$ and $\mathcal{H}(\mathbf{X})$, respectively.
  2. (**Natural inclusion**) Let $\underline{f}^* = \max\left\{\underline{f}^*, f(\mathbf{a})\right\}$ and $M(\mathbf{X}) = \max(f(\mathbf{X}))$. If $M(\mathbf{X}) \leq \underline{f}^*$ then return.
  3. (**Monotonicity test**) If $0 \notin \nabla f(\mathbf{X})$ then let $M(\mathbf{X}) = -\infty$ and return.
  4. (**First order Taylor inclusion**) Let

  $$M(\mathbf{X}^*) = \min\left\{M(\mathbf{X}), \max\left(f(\mathbf{a}) + \nabla f(\mathbf{X})^T(\mathbf{X} - \mathbf{a})\right)\right\}.$$

  If $M(\mathbf{X}^*) \leq \underline{f}^*$ then return.
  5. (**Fast convexity test**) If $\mathcal{H}_{ii}(\mathbf{X}) > 0$ for any $i \in \{1, \ldots, n\}$ then let $M(\mathbf{X}) = -\infty$ and return.
  6. (**Modified second order Taylor inclusion**) Let

  $$M(\mathbf{X}) = \min\left\{M(\mathbf{X}), \max\left(f(\mathbf{a}) - (\mathbf{X} - \mathbf{a})^T\mathcal{H}(\mathbf{X})(\mathbf{X} - \mathbf{a})/2\right)\right\}.$$

  If $M(\mathbf{X}) \leq \underline{f}^*$ then return.
  7. (**Bounding Hessian evaluation**) Compute the upper and lower Hessians of f restricted to $\mathbf{X}$ associated with the interval Hessian $H(\mathbf{X})$, $U(\mathbf{X})$ and $L(\mathbf{X})$ respectively.
  8. (**Convexity Test**) If $L(\mathbf{X})$ is not negative semi-definite then let $M(\mathbf{X}) = -\infty$ and return.

9. (**Bounding Hessian inclusion**) Let y be the maximum of $\nabla f(\mathbf{a})^T + (\mathbf{x}-\mathbf{a})^T L(\mathbf{X})(\mathbf{x}-\mathbf{a})/2$. Let $\underline{f}^* = \max\{\underline{f}^*, y\}$ and

$$M(\mathbf{X}) = \min\left\{M(\mathbf{X}), \max\left(y + (\mathbf{x}-\mathbf{a})^T(U(\mathbf{X}) - L(\mathbf{X}))(\mathbf{x}-\mathbf{a})/2\right)\right\}$$

and return.

- (**Subdivision Rule**) Trisect along the longest edge and evaluate the function and gradient at the mid-points of the outer boxes.
- (**Choosing Rule**) If memory is not full then choose the best subdomain (that is, greatest associated upper bound), otherwise choose the youngest subdomain with associated upper bound greater than the current lower bound $+\epsilon$.

5.4.1 Comments

While the algorithm described is suitable only for objective functions defined on (unions of) boxes, it can be extended to other domains in two ways. Firstly, if the domain is contained in a box and a test can be performed to determine whether any (small) subbox is contained in the domain or it's complement then the full branch and bound framework could be used. The reader is referred to [32] for the details.

Alternatively, if generalised interval arithmetic is used then it is sometimes possible to start with the initial domain and use direct modifications of the steps in the bounding rule. In particular, if inclusion functions over the set of all simplexes are known for the function, gradient and Hessian and the domain is a polytope or a simplex then the natural generalised interval arithmetic inclusion function and the monotonicity test apply. Also, all of the methods using bounding Hessians could be used, if desired, to find upper and lower bounds on the global maximum restricted to any simplex. Thus, a cubic order of convergence in the bounding rule could be achieved on the class of all simplexes. Simplexes can be subdivided along the longest edge as described above.

None of the bounding rule steps described supersedes any other. In other words, examples can be found for which any of the steps provides the best bound. The ordering of the bounding rule steps reflects the amount of work required.

The bounding rules which used only upper Hessians or only lower Hessians discussed above are left out because they cannot be used efficiently on boxes. The

modified second order Taylor inclusion supersedes the inclusion given by the second order Taylor form and so is used instead.

The evaluation of the interval gradients and Hessians in the bounding rule are performed in the first step but, subsequently, may not be needed. It might prove worthwhile, although it has not been tested, to put off the computation of the interval gradient and Hessian until they are needed. However, if automatic differentiation is used then interval inclusions for the function, gradient and Hessian are computed in parallel. If the evaluation of the gradient and Hessian is delayed then this parallel computation is not taken advantage of.

Any of the bounding rule steps can be skipped (with obvious restrictions). Also, if the global maximum can be bounded by any other rule then those rules can be added. Worthwhile rules might include a local maximisation and an interval Newton step as used, for instance, by Hansen [27]. In Chapter 6 skipping the modified second order Taylor inclusion step and the steps associated with using bounding Hessians is investigated empirically.

Finally, rather than always computing every step of the bounding rule, it might be better to switch between them according to some rule. For instance, Ratschek and Rokne suggest using only the natural inclusion if the width of $\mathbf{X}$ is greater than $1/n$ [59]. As long as the cubicly convergent rule is eventually chosen the bounding rule will remain cubicly convergent.

5.5 On Orders of Convergence

This section presents some theoretical implications of achieving a cubic order convergence.

The first result in this section, due to Ratschek and Rokne [56, Theorem 6, page 106], assumes the initial domain D_0 is a box. The algorithm they consider is the branch and bound algorithm, due to Moore and Skelboe [63, 44]. The subdivision rule bisects along the longest edge of the given box. The choosing rule is width-first. The bounding rule is left undefined but gives upper and lower bounds on the global maximum of f restricted to $\mathbf{X}$, $M_f(\mathbf{X})$ and $m_f(\mathbf{X})$ respectively. It is assumed to have the inclusion monotonicity property: $\mathbf{X} \subseteq \mathbf{Y}$ implies $[m_f(\mathbf{X}), M_f(\mathbf{X})] \subseteq [m_f(\mathbf{Y}), M_f(\mathbf{Y})]$ for all boxes $\mathbf{X}, \mathbf{Y} \subseteq D_0$. The collection of boxes at the kth iteration

of this algorithm is denoted $\mathcal{B}_k$. With these assumptions the result can now be stated.

Theorem 5.5.1 (Ratschek and Rokne 1988) *Take $f : D_0 \in \mathbb{R}$, where D_0 is a box in $\mathbb{R}^n$ and suppose that $M_f(\mathbf{X}) - \max\{f(\mathbf{x})|\mathbf{x} \in \mathbf{X}\} \leq Cw(\mathbf{X})^\alpha$, for all boxes $\mathbf{X} \subseteq D_0$ and some constants $C, \alpha > 0$. Then*

$$\max\{M_f(\mathbf{X})|\mathbf{X} \in \mathcal{B}_k\} - \max\{f(\mathbf{x})|\mathbf{x} \in D_0\} \leq C(2w)^\alpha (k+1)^{-\alpha/n},$$

where w is the width of D_0, for all $k \geq 1$.

Clearly, the inclusion monotonicity property can be weakened by quantifying only over the boxes generated by the algorithm and the theorem will still hold.

The bound on the *error* of the function given by the above theorem is of little practical use since the constant C is in general unknown and it says nothing about the variation. However, by using Ratschek and Rokne's Theorem the following more useful result can be proven.

Corollary 5.5.1 *Using the same definitions as in Theorem 5.5.1, suppose that $M_f(\mathbf{X}) - m_f(\mathbf{X}) = O(w(\mathbf{X})^\alpha)$ for all $\mathbf{X} \subseteq D_0$. Then, given $\epsilon > 0$, $O(\epsilon^{-n/\alpha})$ iterations are required to find the global maximum to within ϵ.*

Proof: Fix $f : D_0 \to \mathbb{R}$ and $k \geq 1$. Let $g : D_0 \to \mathbb{R}$ be defined by

$$g(\mathbf{x}) = \max\left\{ m(\mathbf{X})|\mathbf{x} \in \mathbf{X} \text{ and } \mathbf{X} \in \bigcup_{i=1}^{k} \mathcal{B}_i \right\}.$$

Note that, $\max\{g(\mathbf{x})|\mathbf{x} \in D_0\} = \max\{m_f(\mathbf{X})|\mathbf{X} \in \mathcal{B}_k\}$. Define a bounding rule for g as follows: let $M_g(\mathbf{X}) = M_f(\mathbf{X})$ and $m_g(\mathbf{X}) = m_f(\mathbf{X})$ if $\mathbf{X} \in \bigcup_{i=1}^{k} \mathcal{B}_i$ and $M_g(\mathbf{X}) = m_g(\mathbf{X}) = \max\{g(\mathbf{x})|\mathbf{x} \in \mathbf{X}\}$ otherwise. Hence, $M_g(\mathbf{X}) - m_g(\mathbf{X}) \leq M_f(\mathbf{X}) - m_f(\mathbf{X})$, $\forall \mathbf{X} \subseteq D_0$.

Since $M_f(\mathbf{X}) - m_f(\mathbf{X}) \leq Cw(\mathbf{X})^\alpha$ for some $C > 0$ and all $\mathbf{X} \subseteq D_0$, it follows that $M_g(\mathbf{X}) - \max\{g(\mathbf{x})|\mathbf{x} \in \mathbf{X}\} \leq M_g(\mathbf{X}) - m_g(\mathbf{X}) \leq Cw(\mathbf{X})^\alpha$ for all $\mathbf{X} \in D_0$. Now,

$$\begin{aligned} & \max\{M_f(\mathbf{X})|\mathbf{X} \in \mathcal{B}_k\} - \max\{m_f(\mathbf{X})|\mathbf{X} \in \mathcal{B}_k\} \\ = \; & \max\{M_g(\mathbf{X})|\mathbf{X} \in \mathcal{B}_k\} - \max\{g(\mathbf{x})|\mathbf{x} \in D_0\} \\ \leq \; & C(2w)^\alpha (k+1)^{-\alpha/n} \end{aligned}$$

by the Theorem. Since C,w,α and n are independent of k, the variation at the kth iteration $\epsilon(k) \leq C'k^{-\alpha/n}$, for some constant $C' > 0$.

Given $\epsilon > 0$, $\epsilon(k) \leq C'k^{-\alpha/n} \leq \epsilon$, for all $k > (\epsilon/C')^{-n/\alpha}$. Thus the number of iterations required to bound the global maximum to within ϵ is $O(\epsilon^{-n/\alpha})$. □

Thus in the worst case and for sufficiently small ϵ, a cubicly convergent ($\alpha = 3$) bounding rule leads to fewer iterations compared with a bounding rule that is only linearly ($\alpha = 1$) or quadratically ($\alpha = 2$) convergent.

The algorithm in the previous section has a cubic order of convergence. While the condition $\mathbf{X} \subseteq \mathbf{Y}$ implies $[m(\mathbf{X}), M(\mathbf{X})] \subseteq [m(\mathbf{Y}), M(\mathbf{Y})]$ has not been proven for the bounding rule, it could easily be enforced on all boxes the algorithm generates, if required.

Corollary 5.5.2 *Given $f : D_0 \to \mathbb{R}$ and $\epsilon > 0$, a branch and bound algorithm with the bounding rule of Section 5.4 with enforced inclusion monotonicity on generated boxes, a bisection along the longest edge subdivision rule and a width-first choosing rule, requires at most $O(\epsilon^{-n/3})$ iterations to find the global maximum to within ϵ.*

The subdivision rule of Section 5.4 is similar except that, instead of bisecting, it trisects the longest edge. The choosing rule is the same as long as the priority queue does not become full.

Conjecture 5.5.1 *The algorithm presented in the previous section requires $O(\epsilon^{-n/3})$ iterations in the worst case.*

In practice, the average case complexity is of more relevance since the worst case hardly ever occurs. Also, since ϵ is limited by the accuracy of the machine, results that apply for "sufficiently small" ϵ may never be realised.

Another important factor in the performance of branch and bound algorithms is how quickly boxes are removed by being fathomed compared to how quickly they are generated by subdivision. Ratschek and Rokne claim that with a width-first choosing rule, since $\bigcap_{i=1}^{\infty} \cup \mathcal{B}_k = X^*$ and at most 2^n boxes with disjoint interiors can contain the same point, the number of boxes in the collection will eventually always be less than $s2^n$, where s is the number of global maximizers [56, Corollary 1 of Theorem 8]. This conclusion is incorrect as shown by the following example.

Counter Example 5.5.1 *Let $f : [-1, 1] \to \mathbb{R}$ be defined by $f(x) = -x^2$ and the bounding rule be defined by $M([a, b]) = \max\{f(x) | x \in [a, b]\}$ and $m([a, b]) =$*

$M([a,b]) - 2(b-a)$. *Then with a width-first choosing rule and a bisection subdivision rule, the number of boxes in the collection is unbounded.*

Proof: Fix any $m > 1$. Since the subdivision bisects the intervals in the collection and the choosing rule is width-first, given any $m > 1$, at some stage all the intervals in the collection will be of the form $[i/2^m, (i+1)/2^m]$, for some integer i and all have width $1/2^m$. The maximum lower bound on the global maximum will be $-1/2^{m-1}$ given by the interval $[0, 1/2^m]$. Thus all the intervals with an upper bound on the global maximum not less than $-1/2^{m-1}$ will be unfathomed. These intervals are $[i/2^m, (i+1)/2^m]$, where $-2^{(m+1)/2} - 1 \leq i \leq 2^{(m+1)/2}$. Thus the number of unfathomed intervals is unbounded. See figure 5.4. □

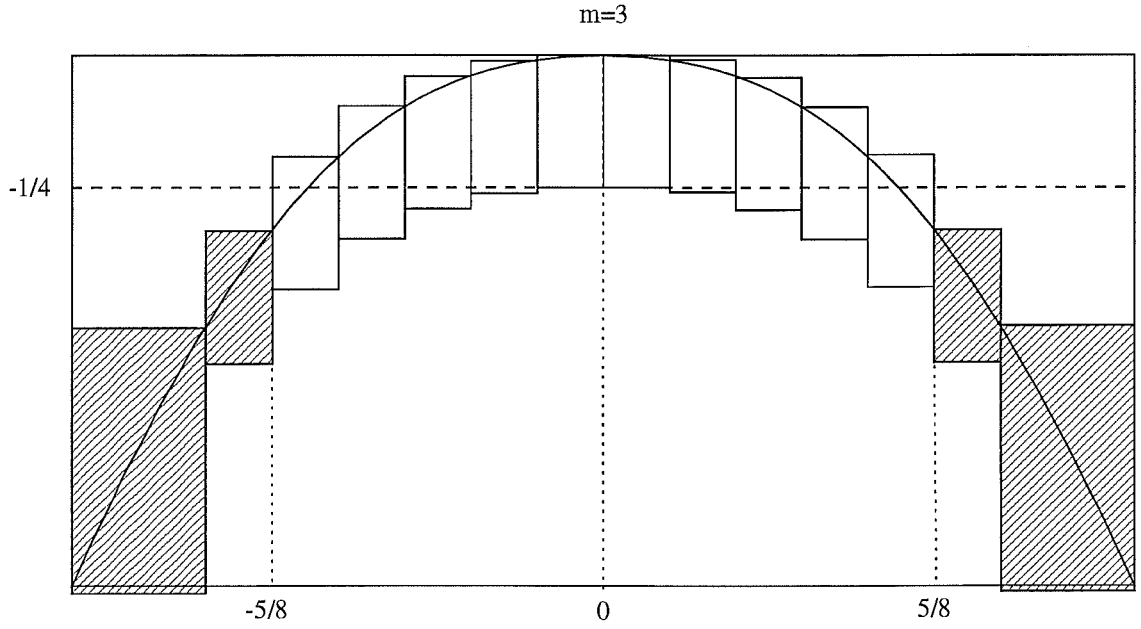

Figure 5.4: The unfathomed boxes (unshaded) on the function $-x^2$ for $m = 3$ from the proof of the counter-example.

The number of boxes in the collection as k tends to infinity depends on the shape of the function around the global optimizers and the bounds given by the bounding rule. If the bounds are "good" for small boxes, then the number of unfathomed boxes can remain finite and small as k tends to infinity. The bounds given by a cubicly convergent bounding rule are guaranteed to be tighter than those of only a quadratic or linearly convergent bounding rule for all sufficiently small boxes. However, even with cubic convergence, it cannot be guaranteed that the number of unfathomed boxes will not tend to infinity.

Chapter 6

Empirical Testing of Bounding Rules

This chapter empirically investigates the adaptive second derivative bounding rule presented in the previous chapter. To this end, several algorithms using adaptive second derivative bounding rules, and one using only first derivatives, were implemented in the branch and bound framework with the choosing and subdivision rules presented in the previous chapter, and run on test functions.

From a strictly mathematical point of view, empirical results should be viewed with caution. In general the performance of a global optimization algorithm is very dependent on the geometry of the objective function. Algorithms based on interval arithmetic are sensitive even to how the function is expressed. Standard test functions often have very nice properties and are easily expressed. This may not be the case for a particular real-world global optimization application.

It is common practice to compare very different algorithms empirically. Such comparisons would be reasonable if the different algorithms were all applied to solve the same application and were tested on representative examples. However, without an application in mind, empirical tests on an essentially arbitrary set of test functions can be misleading. For instance, an algorithm designed to work on a large class of functions will normally compare poorly to an algorithm which works only on a much smaller class if they are tested on functions from the smaller class; an algorithm which halts when the sample value is within ϵ of the (known) global optimum will normally compare poorly with an algorithm which halts when a guaranteed bound

on the (unknown) global optimum is achieved; an algorithm designed to work on simplicial domains will normally compare poorly with an algorithm designed to work on box domains when tested only on test functions on box domains.

In this thesis, the algorithms compared all have the same stopping rule and largely the same implementation. All that differs is that some of the steps in the bounding rule are dropped.

6.1 The Implementation

The implementation of the algorithm is quite extensive—over 50Kb of C++ code. The main routines are listed in Appendix B. The full source code used for the tests is available from the author upon request.

The algorithm was implemented in C++ primarily to take advantage of operator overloading for interval arithmetic, linear algebra and automatic differentiation. Operator overloading allows, for instance, the arithmetic operators to be defined to act on intervals, returning correct interval inclusions. Together with templates, the same function can take a real vector and return a real number, an interval vector and return a natural inclusion or an interval vector and return interval inclusions of the function, gradient and Hessian (computed in parallel).

The Profil/BIAS library was used for the implementation of interval arithmetic and automatic differentiation as well as most of the linear algebra. A major inefficiency of this library is that symmetric matrices (and symmetric interval matrices) are treated in their full form, requiring n^2 storage and time for calculations. Taking advantage of the symmetry would reduce this storage and time to $n(n+1)/2$ and could lead to significant savings for methods which use (interval) Hessians.

The CLAPACK and NAG libraries were used for negative definite testing and the quadratic programming respectively.

The object oriented nature of C++ was used to implement fairly general routines. For instance, the simplified branch and bound framework was implemented to work on a virtual class called `DomainBound`. This class has virtual member functions for bounding and subdividing a domain. The bounding rule described in the previous chapter was implemented as a type of `DomainBound`, with private storage of the box, together with the function and gradient value at the mid-point. The subdivision rule

passed this information on to its children.

The collection of unfathomed `DomainBounds` is managed by a class with member functions to store, choose and delete fathomed `DomainBounds`. The choosing rule presented in the previous chapter was implemented using a skiplist and a linear list for best-first and depth-first retrieval, respectively.

There are many areas in which the algorithm could be made faster (sometimes at the cost of simplicity and generality). These inefficiencies are at least consistent and the implementation is sufficient for comparing second derivative bounding rules.

6.2 Test Results for Bounding Rules

6.2.1 The Bounding Rules

Four different bounding rules were implemented. The first does not use second derivatives at all, but just the natural inclusion, the monotonicity test and the first order Taylor inclusion. The second uses the fast convexity test and the modified second order Taylor inclusion. The third uses the cubicly convergent bounding Hessian inclusion in addition to the bounds of the first algorithm. The fourth bounding rule uses all of the steps of the algorithm presented in Chapter 5.

Some empirical results for global optimization algorithms in the literature use heuristic modifications which lead to improved performance on the test functions. Even though these modifications are pointed out, the results can be misleading because the modifications are designed for the same functions on which the algorithm is tested. No such modifications have been made to the algorithms presented here.

6.2.2 The Test Functions

The algorithms were run on 29 test functions defined on boxes. The first 28 of these were taken from Hansen [27] (where Hansen proscribes two initial boxes the larger of the two was taken). The test function numbers are the same as his. Test function 19 was not applicable since it is not a real valued function (it is interval valued). Test function 30 is the Peaks function described in Chapter 3.

All of the test functions were programmed in the expression presented (with summation signs expanded out in full). The (interval) automatic differentiation

class in the Profil/BIAS library was used to find (inclusions of) the gradient and Hessian.

A relatively modest tolerance of 10 significant figures or 10 decimal places if $|f(x)| < 1$ was chosen. The global maximum was determined to with 0.1 and these figures used to determine the final tolerance. The quadratic programming subproblem was solved using the NAG library returns an estimate of unknown guaranteed accuracy. The error from this and other sources is expected to be far less than the required tolerance (all internal arithmetic was computed to double precision, approximately 16 significant figures). In all but one case the algorithms which completed returned correct results.

6.2.3 The Results

The CPU times and number of iterations are given in Table 6.1. The CPU times were very close to the elapsed times in all cases where the algorithm completed since virtual memory was not required. These times are the ultimate measure of an algorithm's performance but are dependent on the platform and implementation.

The number of functions, gradient and Hessian evaluations can be computed from the number of iterations. In particular, the first algorithm requires one function evaluation, one interval function and one gradient evaluation per iteration. The second requires an additional interval Hessian evaluation, and the third and forth require a additional gradient evaluation.

6.2.4 Observations

The algorithms using adaptive second derivatives consistently used fewer iterations than the first algorithm which does not use second derivatives. However, the overheads of using second derivatives were such that the CPU times for the adaptive second derivative algorithms were greater on the whole.

Of the algorithms using adaptive second derivatives, the algorithm using only bounding Hessians used fewer iterations than the one using the modified second order Taylor inclusion. The algorithm using both of these rules consistently took the least number of iterations, as expected. The best CPU times are shared fairly evenly between this algorithm and the one using only the modified second order

Algorithm		None		Modified Taylor		Bounding Hessian		Both	
Test	n	CPU	Iter.	CPU	Iter.	CPU	Iter.	CPU	Iter.
1	2	0.27s	312	0.51s	206	0.70s	190	0.65s	184
2	1	0.10s	146	0.05s	32	0.06s	28	0.05s	24
3	1	0.22s	50	0.25s	48	0.24s	38	0.22s	38
4	2	10.1s	1038	13.7s	1013	13.6s	876	12.7s	876
5	2	1.42s	159	2.02s	155	2.28s	145	2.14s	145
6	3	0.21s	37	0.36s	37	0.31s	25	0.28s	25
7	4	0.38s	49	0.69s	49	0.56s	33	0.53s	33
8	5	0.60s	61	1.17s	61	0.96s	41	0.92s	41
9	8	1.64s	97	4.04s	97	3.15s	65	3.00s	65
10	10	2.50s	121	7.61s	121	5.70s	81	5.67s	81
11	2	0.13s	47	0.24s	45	0.23s	36	0.21s	36
12	3	1.64s	122	0.94s	118	1.12s	105	1.00s	105
13	4	2.50s	376	4.31s	370	5.07s	353	4.59s	354
14	5	13.1s	1788	28.2s	1704	34.5s	1767	32.2s	1687
15	7	171s	17022	461s	16588	542s	16993	584s	16565
16	50	—	—	—	—	3.70s**	1**	4.92s**	1**
17	2	0.16s	146	0.49s	128	0.37s	71	0.37s	71
18	3	0.11s	79	0.38s	79	0.36s	64	0.35s	64
19	N/A								
20	2	0.03s	56	0.08s	56	0.00s	1	0.00s	1
21	3	0.92s	86	1.88s	86	2.12s	83	2.10s	83
22	4	—	—	8656s	300521	1331s	41753	1393s	41490
23	4	0.16s	132	0.51s	132	0.59s	102	0.64s	102
24	2	0.34s	718	0.22s	180	0.00s	1	0.00s	1
25	5	0.03s	11	0.14s	11	0.18s	11	0.16s	11
26	10	0.12s	21	0.95s	21	1.17s	21	1.36	21
27	30	0.01s	1	0.25s	1	0.27s	1	0.38	1
28	3	0.04s	37	0.17s	37	0.15s	26	0.15	26
29	2	0.04s	74	0.07s	74	0.12s	63	0.13	63
30	2	0.86s	340	0.66s	103	0.88s	101	0.78	96

Table 6.1: CPU times for adaptive second derivative branch and bound algorithms run on an Ultra Sparc II with 300Mb of memory. A dash indicates the algorithm failed to find the global maximum to within the desired tolerance in reasonable time (24 hours). A double asterisk indicates the algorithm returned incorrect results.

Taylor inclusion.

For all of the algorithms the number of final boxes was fairly low on most of the test functions and often only one box was left. This indicates that for these bounding rules nearly as many boxes are fathomed in each iteration as are created by subdivision (on average).

One notable exception to the overall trend was on the hardest of the test functions, Test Function 22. The first algorithm did not solve the problem within 24 hours. After 19 hours and 7×10^7 iterations, it had found the global optimum to 8 decimal places and had more than 6×10^6 unfathomed boxes. The second algorithm took nearly two and half hours to solve the problem to 10 decimal places and had 16175 remaining boxes. The final two algorithms, which use bounding Hessians both took less than 25 minutes to solve the problem and had only 53 remaining boxes. Thus the overheads associated with computing the cubicly convergent bounds from bounding Hessians pays off in this case.

Finally, Test Function 16 is in fact a 50 dimensional quadratic. The algorithms using the bounding Hessians bounding rule returned incorrect results because the NAG quadratic programming routine gave a result accurate to only two significant figures, far short of the required 10. On quadratic functions (also Test functions 20 and 24) an algorithm using the bounding Hessian rule will only ever perform one iteration (assuming the interval Hessian has width zero). The other two algorithms failed to return in reasonable time. (With 50 dimensions even reducing the width of the initial box would take 3^{50} iterations if no boxes of maximum width are ever fathomed.)

6.3 Conclusions

The greater CPU times for the adaptive second derivative algorithms compared with the first algorithm, for most of the test functions, can in some part be attributed to the inefficiency in the Profil/BIAS library. Also, since there are more symmetric matrix computations for the algorithm using both modified second order Taylor inclusions and bounding Hessians, the overall best CPU time of the adaptive second derivative algorithms is likely to be tipped in favour of this algorithm if these inefficiencies were corrected.

The computation times for the algorithms using adaptive second derivative bounds could further be improved by replacing the automatic differentiation of the Profil/BIAS library used to find interval Hessians, with the rather ingenious method due to Hansen [22]. Hansen's method replaces some of the interval computations with real computations, still providing valid bounds, and is both faster and gives tighter inclusions [22].

Even so, it seems worthwhile to try to avoid the overheads associated with computing interval and bounding Hessians when they are not going to give improved bounds. One way of achieving this could be to occasionally (say 1% of the time) compute the interval and bounding Hessians and associated bounds. Should this lead to improved bounds on a box then bounding Hessians could be computed for all its descendants based on the heuristic justification that the adaptive second derivative method is likely to give improved bounds for all boxes of smaller width in this region. The author is confident that the above improvements to the adaptive second derivative methods will lead to favourable CPU times on the test functions.

Finally, the performance of the algorithm using bounding Hessians on Test Function 22 shows that bounding Hessians are a useful tool for global optimization. Although, given a global optimization problem, it is not known whether the use of bounding Hessians will lead to being able to solve the problem in reasonable time, if all other methods fail, then possibly using bounding Hessians will enable it to be solved.

Chapter 7

Summary and Conclusions

As we have seen, the definition of the global optimization problem is deceptively simple. While some of the ideas are intuitive and simple, the theory can be deep and complex. In practice, solving global optimization problems is usually computationally expensive.

Despite the fact that there are theoretically sound global optimization algorithms, they are hardly ever used in practice. Largely this is due to the fact that most practical problems that are phrased as global optimization problems are really "non-local optimization" problems. Because global optimization algorithms at present are usually much slower than "non-local optimization" algorithms, the latter is and should be used to solve such problems. Global optimization algorithms are useful for problems for which guaranteeing the global optimum is necessary to solve the problem.

All of the global optimization algorithms surveyed and presented require some form of global information about the function. The explanation for this is given in Chapter 2—without global information no finitely terminating algorithm, deterministic or stochastic, can solve (relaxed) global optimization problems.

The function's expression is one such type of global information and is potentially the most powerful since it contains all the information about the function there is to be known. Interval arithmetic can be used to extract useful information, specifically upper and lower bounds on the function restricted to any box, from the function's expression. Generalised interval arithmetic extends this idea and can be used to find upper and lower bounds over other classes of domains. More

importantly, generalised interval arithmetic provides a way to make use of structure in the function's expression. Using such structure could potentially lead to otherwise unsolved problems being solved.

As well as providing bounds on function values, (generalised) interval arithmetic can also be used to provide bounds on gradients and Hessians. The relationship between interval Hessians so obtained and bounding Hessians was examined in Chapter 4. The main result of that chapter gives an optimal method of obtaining non-dominated bounding Hessians of minimum trace from interval Hessians. Bounding Hessians are useful for a number of existing global optimization algorithms.

In this thesis, bounding Hessians, and the method for obtaining them, were used to achieve a cubicly convergent bounding rule for a branch and bound global optimization algorithm. An implementation of this adaptive second derivative algorithm was able to solve all but one of the test functions attempted and gave a considerable improvement on one of these.

References

[1] C. S. Adjiman and C. A. Floudas. Rigorous convex underestimators for general twice-differentiable problems. *Journal of Global Optimization*, 9:23–40, 1996.

[2] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, 1983.

[3] F. Aluffi-Pentini, V. Parisi, and F. Zirilli. Global optimization and stochastic differential equations. *Journal of Optimization: Theory and Applications*, 47:1–16, 1985.

[4] I. P. Androulakis, C. D. Maranas, and C. A. Floudas. αBB: A global optimization method for general constained nonconvex problems. *Journal of Global Optimization*, 7:337–363, 1995.

[5] W. P. Baritompa. Customizing methods for global optimization—a geometric viewpoint. *Journal of Global Optimization*, 3:193–212, 1993.

[6] W. P. Baritompa and A. Cutler. Accelerations for global optimization covering methods using second derivatives. *Journal of Global Optimization*, 4:329–341, 1994.

[7] I. M. Bomze, T. Csendes, R. Horst, and P. Pardalos, editors. *Developments in Global Optimization.* Kluwer Academic Publishers, 1997.

[8] L. Breiman and A. Cutler. A deterministic algorithm for global optimization. *Mathematical Programming*, 58:179–199, 1993.

[9] R. P. Brent. *Algorithms for Minimization without Derivatives.* Prentice-Hall, 1973.

[10] C. L. Brocks III, M. Karplus, and B. M. Pettitt. *Proteins: A Theoretical Perspective of Dynamics, Structure, and Thermodynamics.* John Wiley & Sons, 1988.

[11] U. Burkert and N. L. Allinger. *Molecular Mechanics.* American Chemical Society, Washington, D.C., 1982.

[12] Chew S.-H. and Zheng Q. *Integral Global Optimization: Theory, Implementations and Applications.* Springer-Verlag, 1988.

[13] R. J. Duffin, E. L. Peterson, and C. Zener. *Geometric Programming: Theory and Application.* John Wiley & Sons, 1967.

[14] P. C. Duong. Finding the global extremum of a polynomial function. In *Essays on Nonlinear Analysis and Optimization Problems*, pages 111–120. Institute of Mathematics, Hanoi, Vietnam, 1987.

[15] J. Falk. Conditions for global optimality in nonlinear programming. *Operations Research*, 21:337–340, 1973.

[16] C. A. Floudas and P. M. Pardalos, editors. *State of the Art in Global Optimization.* Kluwer Academic Publishers, 1995.

[17] C. A. Floudas and V. Visweswaran. A global optimization algorithm (GOP) for certain classes of nonconvex NLPs—I. Theory. *Computers and Chemical Engineering*, 14:1397–1417, 1990.

[18] K. Fukami and Y. Tateno. On the uniqueness of the maximum likelihood method for the estimating molecular trees: Uniqueness of the likelihood point. *Journal of Molecular Evolution*, 31:511–523, 1989.

[19] S. Geman and Hwang C.-R. Diffusions for global optimization. *SIAM Journal of Control and Optimization*, 24:1031–1043, 1986.

[20] D. Goldfarb and S. Liu. An $O(n^3L)$ primal interior point algorithm for convex quadratic programming. *Mathematical Programming*, 49:325–340, 1988.

[21] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13:311–329, 1988.

[22] E. R. Hansen. On solving systems of equations using interval arithmetic. *Mathematics of Computing*, 22:374–384, 1968.

[23] E. R. Hansen. A generalized interval arithmetic. In K. L. Nickel, editor, *Interval Mathematics*, pages 7–18. Springer-Verlag, 1975.

[24] E. R. Hansen. Global optimization using interval analysis—the multidimensional case. *Numerische Mathematik*, 34:247–270, 1980.

[25] E. R. Hansen. *Global optimization using interval analysis.* Marcel Dekker, 1992.

[26] P. Hansen and B. Jaumard. Lipschitz optimization. In Horst and Pardalos [32], pages 407–493.

[27] P. Hansen, B. Jaumard, and S.-H. Lu. Global optimization of uni-variate Lipschitz functions: II. New algorithms and computational comparison. *Mathematical Programming*, 55:273–292, 1992.

[28] P. Hansen, B. Jaumard, and Lu S.-H. On using estimates of Lipschitz constants in global optimization. *Journal of Optimization Theory and Applications*, 75:195–200, 1992.

[29] J. Hass, M. Hutchings, and R. Schlafly. The double bubble conjecture. *ERA American Mathematics Society*, 1:98–102, 1995.

[30] J. G. Hocking and G. S. Young. *Topology.* Addison-Wesley Publishing Company, 1961.

[31] A. J. Hoffman and P. Wolfe. History. In Lawler et al. [38], pages 1–16.

[32] R. Horst and P. M. Pardalos, editors. *Handbook of Global Optimization.* Kluwer Academic Publishers, 1995.

[33] R. Horst and H. Tuy. *Global optimization: Deterministic Approaches.* Springer-Verlag, second, revised edition, 1993.

[34] K. Ichida and Y. Fujii. An interval arithmetic method for global optimization. *Computing*, 23:85–97, 1979.

[35] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization: Theory and Application*, 79:157–181, 1993.

[36] B. B. Kearfott. Preconditioners for the Guass-Seidel method. *SIAM Journal on Numerical Analysis*, 27:804–822, 1990.

[37] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:621–680, 1983.

[38] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem.* John Wiley & Sons, 1985.

[39] W. B. Liu and C. A. Floudas. A remark on the GOP algorithm for global optimization. *Journal of Global Optimization*, 3:519–521, 1993.

[40] D. MacLagan, T. Sturge, and W. Baritompa. Equivalent methods for global optimization. In Floudas and Pardalos [16], pages 201–211.

[41] N. Metropolis, J. Howlett, and G.-C. Rota, editors. *A History of Computing in the Twentieth Century.* Academic Press, 1980.

[42] R. H. Mladineo. An algorithm for finding the global maximum of multimodal, multivariate functions. *Mathematical Programming*, 34:188–200, 1986.

[43] R. E. Moore. *Interval Analysis.* Series in Automatic Computation. Prentice-Hall, 1966.

[44] R. E. Moore. On computing the range of values of a rational function of n variables over a bounded region. *Computing*, 16:1–15, 1976.

[45] R. E. Moore, editor. *Reliability in Computing: The Role of Interval Methods in Scientific Computing.* Academic Press, 1988.

[46] R. Murray and C. P. Stephens. A depth-first Lipschitz based global maximization algorithm. Technical Report 102, Department of Mathematics and Statistics, University of Canterbury, Christchurch, New Zealand, 1993.

[47] A. Neumaier. The enclosure of solutions of parameter-dependent systems of equations. In Moore [45], pages 269–286.

[48] A. Neumaier. *Interval Methods for Systems of Equations.* Cambridge Press, 1990.

[49] A. Neumaier. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM Review*, 39:407–460, 1997.

[50] Numerical Algorithms Group Limited. *The NAG Fortan Library Manual, Mark 14*, 1990.

[51] P. Pardalos and G. Xue, editors. *Advances in Computational Chemistry and Protein Folding*, volume 4, 1994. Special Issue of the Journal of Global Optimization.

[52] P. Pardalos and G. Xue, editors. *Advances in Computational Chemistry and Protein Folding*, volume 11, 1997. Special Issue of the Journal of Global Optimization.

[53] M. Pinkus. A closed form solution of certain programming problems. *Operations Research*, 16:690–694, 1968.

[54] S. A. Piyavskii. An algorithm for finding the absolute extremum of a function. *USSR Computational Mathematics and Mathematical Physics*, 12:57–67, 1972.

[55] S. B. Prusiner. The prion diseases. *Scientific American*, pages 30–37, Jan 1995.

[56] H. Ratcheck and J. Rokne. *New computer methods for global optimization.* Ellis Horwood, 1988.

[57] H. Ratschek and J. Rokne. *Computer methods for the range of functions.* Ellis Horwood, 1984.

[58] H. Ratschek and J. Rokne. The transistor modeling problem again. *Microelectronics and Reliability*, 32:1725–1740, 1992.

[59] H. Ratschek and J. Rokne. Interval methods. In Horst and Pardalos [32], pages 751–828.

[60] F. Scheon. On a sequential search strategy in global optimization problems. *Calcolo*, 19:321–334, 1982.

[61] Ya. D. Sergeyev. A method using local tuning for minimizing functions with Lipschitz derivatives. In Bomze et al. [7], pages 199–215.

[62] B. O. Shubert. A sequential method seeking the global maximum of a function. *SIAM Journal on Numerical Analysis*, 9:379–388, 1972.

[63] S. Skelboe. Computation of rational interval functions. *BIT*, 14:87–95, 1974.

[64] F. J. Solis and R. J.-B. Wets. Minimization by random search techniques. *Mathematics of Operation Research*, 6:19–30, 1981.

[65] M. A. Steel. The maximum likelihood point for a phylogenetic tree is not unique. *Systematic Biology*, 43:560–564, 1994.

[66] C. P. Stephens. Interval and bounding Hessians. In Bomze et al. [7], pages 109–119.

[67] R. G. Strongin. On the convergence of an algorithm for finding a global extremum. *Engineering Cybernetics*, 11:549–555, 1973.

[68] A. Törn and A. Žilinskas. *Global Optimization.* Springer-Verlag, 1989.

[69] H. Tuy. D.C. optimization: Theory, methods and algorithms. In Horst and Pardalos [32], pages 149–216.

[70] S. Vavavsis. *Nonlinear Optimization: Compexity Issues.* Oxford University Press, 1991.

[71] J. H. Wilkinson. Turing's work at the National Physical Laboratory. In Metropolis et al. [41], pages 101–122.

[72] G. R. Wood. Multidimensional bisection and global optimization. *Computers and Mathematics with Applications*, 21:161–172, 1991.

[73] Z. B. Zabinsky and R. L. Smith. Pure adaptive search in global optimization. *Mathematical Programming*, 53:323–338, 1997.

[74] Zhang B. P., G. Wood, and W. Baritompa. Multidimensional bisection: The perforamance and the context. *Journal of Global Optimization*, 3:193–212, 1993.

[75] V. S. Zhirov. Searching for a global extremum of a polynomial on a parallelepiped. *USSR Computational Mathematics and Mathematical Physics*, 25:163–180, 1985. (In Russian).

Appendix A

An Unsolved Problem

The function given in Example 3.3.3 of Chapter 3 was suggested by Chris Tuffley (personal communication) as a potential not-so-pathological counter-example to the claims of Fukami and Tateno [18]. Fukami and Tateno claimed that "under simple models of sequence evolution there exists at most one point where the likelihood function for a given phylogentic tree is maximized" [65]. Steel [65] gives a simple, but arguably pathological, counter-example to this claim.

The example was hoped to have multiple global maximizers in the interior. Confirming this can be phrased as a series of global optimization problems: show that there exists $\epsilon > 0$ such that the global maximum to within ϵ of f restricted to the boundary of $[0,1]^5$ is less than $f(\mathbf{x})$, for some $\mathbf{x} \in (0,1)^5$. It was hoped that a global maximum to within ϵ, for some $\epsilon > 0$ could be found that was less than a "good" function value of an interior point found by a "non-local optimization algorithm".

All attempts by the author to find the global maximum of this function to a "reasonable" accuracy have failed. This includes standard interval methods (without interval Newton steps) and the algorithm presented in Chapter 5, with and without generalised interval arithmetic. One possible reason for this is there may be an infinite number of global maximizers. Empirically, it was observed that the number of unfathomed boxes grows very large in all the attempts to maximize the function.

Subsequent empirical investigations suggest the example is unlikely to provide a counter example. For instance, using the `constr` local minimization algorithm of MATLAB on the negative of the function returns only points on the boundary (or very close to it) from hundreds of random starting points. Also, adding weight

to the possibility of an infinite number of global optimizers, many putative local optimizers were found.

In order to find a counter-example, more likelihood functions arising from these simple models should be examined. If an example with two or more interior local maximizers can be found, and no boundary points can be found that have greater function values, then a global optimization algorithm may be able to be used to prove no maximizers are on the boundary.

Appendix B

The Main C++ Routines

B.1 BB.C

The following routine implements the branch and bound framework and is used for the branch and bound algorithm in this thesis. It takes a bracket of the global maximizers `bracket`, a tolerance `eps`, and the maximum number of iterations allowed `iterations`. The bracket is usually initialised to contain only the initial domain (together with bounds) but can be a bracket previously attained. Thus, the routine can start from where it left off. Also, the global variable `lower_bound` is assumed to be initialised to a valid lower bound to the function. On exit, if the bracket is empty then the global maximum was less than `lower_bound` or there were no zero derivative global maximizers in the initial bracket. If the bracket is not empty, the global maximum is less than `lower_bound+eps`, the maximizers are contained in the final bracket, and `iteration` iterations were required.

```
extern REAL lower_bound;

void BB(DomainBoundStore &bracket, REAL &eps , INT &iterations){
  INT i=0;
  REAL upper_bound = bracket.upper_bound();
  LinearList<DomainBound *> new_domains;
  DomainBound *p_current_domain;
```

```
  while ((upper_bound - lower_bound) > eps && i!=iterations
         && !bracket.is_empty()){
    p_current_domain = bracket.choose();

    p_current_domain->subdivide(new_domains);
    delete p_current_domain;

    while (!new_domains.is_empty()){
      p_current_domain = new_domains.retrieve_first();
      p_current_domain->compute_bound();
      if (p_current_domain->upper_bound >= lower_bound)
        bracket.store(p_current_domain);
      else
        delete p_current_domain;
    }

    if (!bracket.is_empty())
      upper_bound = bracket.upper_bound();

    bracket.remove_fathomed(lower_bound);
    i++;
  }
  iterations = i;
  eps = upper_bound-lower_bound;
}
```

B.2 DomainBoundStore.C

The following is the declaration and definition `DomainBoundStore` which is used by `BB` to maintain the bracket of global optimizers. `DomainBoundStore` implements a hybrid depth-first/width-first choosing rule.

```
class DomainBoundStore{
  REAL eps;
```

```
  REAL upper_bound;
  INT pq_max_size;
  SkipList<pDomainBound> pq;
  LinearList<pDomainBound> stack;
public:
  DomainBoundStore(REAL e, INT s);
  ~DomainBoundStore();
  void store(DomainBound *);
  DomainBound *choose();
  void remove_fathomed(REAL f_min);
  REAL upper_bound();
  BOOL is_empty();
  INT size();
};

extern REAL lower_bound;

DomainBoundStore::DomainBoundStore(REAL e,INT s){
  eps = e;
  pq_max_size = s;
  upper_bound = Machine::PosInfinity;
}

DomainBoundStore::~DomainBoundStore(){
  DomainBound *p;
  while (!stack.is_empty()){
    p =  stack.first();
    delete p;
    stack.remove_first();
  }
  while (!pq.is_empty()){
    p= pq.first();
    delete p;
```

```
    pq.remove_first();
  }
}

void DomainBoundStore::store(DomainBound *p){
  if (pq.size()<pq_max_size || p->upper_bound - lower_bound < eps)
    pq.insert(p);
  else
    stack.insert(p);
}

DomainBound *DomainBoundStore::choose(){
  DomainBound *p;
  if (!stack.is_empty()){
    p=stack.first();
    stack.remove_first();
  }
  else {
    p=pq.first();
    pq.remove_first();
  }
  return p;
}

void DomainBoundStore::remove_fathomed(REAL f_min){
  DomainBound *p = pq.last();
  while(!pq.is_empty() && p->upper_bound < f_min){
    pq.remove_last();
    delete p;
    p = pq.last();
  }
}
```

```
REAL DomainBoundStore::upper_bound(){
  // Instead of scanning the stack as well as checking the first
  // DomainBound in the Priority Queue, the last upper_bound when
  // the stack was empty is used.

  if (stack.is_empty())
    upper_bound = ((DomainBound *)pq.first())->upper_bound;
  return upper_bound;
}

BOOL DomainBoundStore::is_empty(){
  return pq.is_empty() && stack.is_empty();
}

INT DomainBoundStore::size(){
  return pq.size() + stack.size();
}
```

B.3 DomainBound.H

The follow is the declaration of the virtual class `DomainBound`. Not that the lower bound is not stored since a global variable to keep the greatest lower bound is used instead. The `BB` routine works on classes of this type which provide the bounding and subdivision rules as member functions. A class of pointers to `DomainBound`, `pDomainBound`, is defined so that the ordering operator > can be defined.

```
class DomainBound{
public:
  REAL upper_bound;
  DomainBound(){}
  virtual ~DomainBound(){};
  virtual void compute_bound() = 0;
  virtual void subdivide(LinearList<DomainBound *> &) const = 0;
};
```

```
class pDomainBound{
  DomainBound *p;
public:
  pDomainBound(){}
  pDomainBound(DomainBound *q){p = q;}
  operator DomainBound *() const {return p;}
  operator DomainBound &() const {return *p;}
};

int operator >(pDomainBound p, pDomainBound q){
  return ((DomainBound *)q)->upper_bound >
                              ((DomainBound *)p)->upper_bound;
}
```

B.4 BoxBoundingHessian.C

The following is the class declaration of `BoxBoundingHessian` which is the type of `DomainBound` used for the new algorithm. It expects the global variables called `evaluate_gradient` and `evaluate_hessian` to point to functions that return a function and gradient value at a point, and an interval function, gradient and Hessian value over a box, respectively. Also, the global variable `lower_bound` is expected to be initialised with a valid lower bound of the global maximum and it updated by the member function `compute_bound`.

```
extern void (* evaluate_gradient)(const VECTOR &x, GRADIENT &y);
extern void (* evaluate_hessian)(const INTERVAL_VECTOR &X,
                                 INTERVAL_AUTODIFF &H);
extern REAL lower_bound;

class BoxBoundingHessian : public DomainBound{
  INTERVAL_VECTOR X;             // The current box.
  GRADIENT y;                    // The function and gradient value
```

```
                                    // at the mid point.
public:
  BoxBoundingHessian(){}
  BoxBoundingHessian(const INTERVAL_VECTOR& X);
  BoxBoundingHessian(const BoxBoundingHessian &);
  ~BoxBoundingHessian(){}
  void subdivide(LinearList<DomainBound *> &) const;
  void compute_bound();
};

BoxBoundingHessian::BoxBoundingHessian(const INTERVAL_VECTOR &X1){
  X = X1;
  (*evaluate_gradient)(Mid(X),y);
}

BoxBoundingHessian::BoxBoundingHessian(const BoxBoundingHessian &D){
  X=D.X;
}


void
BoxBoundingHessian::subdivide(LinearList<DomainBound*> &new_domains)
                                                                  const{
  int k = 1;
  REAL md = Diam (X(1));
  REAL d;

  for (int i = 2; i <= Dimension (X); i++)
    if ((d = Diam(X(i))) > md) {
      k = i;
      md = d;
    }
```

```
  BoxBoundingHessian *p1 = new BoxBoundingHessian(*this);
  BoxBoundingHessian *p2 = new BoxBoundingHessian(*this);
  BoxBoundingHessian *p3 = new BoxBoundingHessian(*this);

  REAL a=Inf(X(k)),b=Sup(X(k));
  p1->X(k) = Hull(a,a+md/3);
  p2->X(k) = Hull(a+md/3,b-md/3);
  p3->X(k) = Hull(b-md/3,b);

  (*evaluate_gradient)(Mid(p1->X),p1->y);
  p2->y = y;
  (*evaluate_gradient)(Mid(p3->X),p3->y);

  new_domains.insert(p1);
  new_domains.insert(p2);
  new_domains.insert(p3);
}


void BoxBoundingHessian::compute_bound(){
  int n=Dimension(X);
  INTERVAL_VECTOR X_minus_a = X-Mid(X);
  REAL local_lower_bound = REAL(y);
  if (local_lower_bound > lower_bound)
    lower_bound = local_lower_bound;

  INTERVAL_AUTODIFF H;
  (*evaluate_hessian)(X,H);

  // Natural Interval Inclusion
  upper_bound = Sup(FunctionValue(H));

  if (upper_bound < lower_bound)
```

```
    return;

  // Zero Derivative Test
  for (int i=1; i<=n; i++){
    if (! (0<=GradientValue(H)(i))){
      upper_bound = Machine::NegInfinity;
      return;
    }
  }

  // First order Taylor inclusion
  REAL upper_bound1 = lower_bound + Sup(GradientValue(H)*(X_minus_a));
  if (upper_bound1 < upper_bound){
    upper_bound = upper_bound1;
    if (upper_bound < lower_bound)
      return;
  }

  // Fast Convexity Check
  for (int i=1; i<=n ;i++){
    if (Inf(HessianValue(H)(i,i)) > 0){
      upper_bound = Machine::NegInfinity;
      return;
    }
  }

  // Modified second order Taylor inclusion
  upper_bound1 = REAL(y)+Sup(X_minus_a*(HessianValue(H)*X_minus_a))/2;
  if (upper_bound1 < upper_bound){
    upper_bound = upper_bound1;
    if (upper_bound < lower_bound)
      return;
  }
```

```
  // Find Bounding Hessians.
  MATRIX L(n,n);
  VECTOR U_minus_L(n);
  IntervalHessian2BoundingHessians(H,L,U_minus_L);

  // Convexity Check
  if (!is_Negative_Semi_Definite(L)){
    upper_bound = Machine::NegInfinity;
    return;
  }

  // Cubic Bounding Hessian Rule
  REAL width = 0; // Difference between upper and lower bound on the
                  // global maximum restrict to D.
  for (int i=1;i<=n;i++)
    width += Sqr(Diam(X(i)/2))*U_minus_L(i)/2;

  VECTOR x(n);
  REAL local_lower_bound = REAL(y)+QP(VECTOR(y),L,X,x);

  if (local_lower_bound > lower_bound)
    lower_bound = local_lower_bound1;

  if (local_lower_bound + width < upper_bound)
    upper_bound = local_lower_bound + width;
}
```

B.5 Misc.C

The following routine implements the method of obtaining bounding Hessians from Interval Hessians. The `compute_bounds` member function of `BoxBoundingHessian` calls it. Rather than return full matrices for both upper and lower Hessians which

differ only on the diagonal, only the lower Hessian and a vector of the differences between the upper and lower Hessian along the diagonal.

```
void IntervalHessian2BoundingHessians(const INTERVAL_MATRIX &H,
                                      MATRIX &L,VECTOR &U_minus_L){
  // This routine could be made more efficient by using the symmetry
  // of the interval matrix.
  L=Mid(H);
  Clear(U_minus_L);
  for (int i=1; i<= RowDimension(H);i++){
    for (int j=1; j<= ColDimension(H);j++)
      U_minus_L(i)+=(Diam(H(i,j))+Diam(H(j,i)))/2;
    L(i,i)-=U_minus_L(i)/2;
  }
}
```